Exercício 5

Arthur Felipe Reis Souza

April 14, 2024

1 Introduction

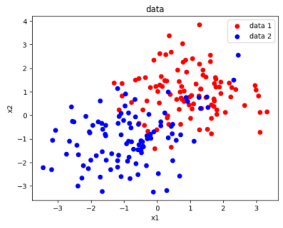
As redes neurais ELM's (Extreme Learning Machines) se baseiam no Teorema de Cover, o qual indica que um problema não linearmente separável pode se tornar linearmente separável através de uma expansão aleatória de alta dimensão da camada intermediária.

Portanto, as redes neurais desse tipo contém 3 camadas, uma de entrada, uma intermediária e uma de saída. A camada intermediária consistirá de um número grande de neurônios (hyperparâmetro p que deve ser determinado) que conterão pesos aleatórios. As redes ELM's, por se basearem em uma expansão aleatória, não conseguem alcançar uma solução ótima. Essa estocásticidade leva às soluções subótimas do problema. É válido destacar que esse tipo de rede fixa os pesos Z da camada intermediária e gera uma projeção das entradas sobre a camada intermediária H(xZ). Essa projeção será a entrada para a camada de saída. A saída da rede será y = Hw, e w é da forma : w = pseu(H)y.

1.1 2dnormals

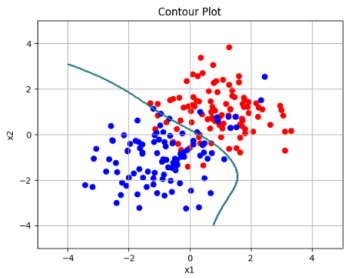
O primeiro problema que será realizado é gerado pelo pacote mlbench. Após as gerações dos dados obtemos o gráfico de amostras :

```
[267]: labels_df_reshaped = labels_df_normals.reshape(-1, 1)
       mat_plot = np.concatenate((x_df_normals, labels_df_reshaped), axis = 1)
       index_sort = 2
       sorted_indices = np.argsort(mat_plot[:, index_sort])
       mat_plot = mat_plot[sorted_indices]

[268]: x1_points = mat_plot[: 99, 0 : 2]
       x2_points = mat_plot[99 :, 0 : 2]
       plt.scatter(x1_points[:, 0], x1_points[:, 1], color = 'red', label = 'data 1')
       plt.scatter(x2_points[:, 0], x2_points[:, 1], color = 'blue', label = 'data 2')
       plt.title('data')
       plt.xlabel('x1')
       plt.ylabel('x2')
       plt.legend()
       plt.plot()

[268]: []
```

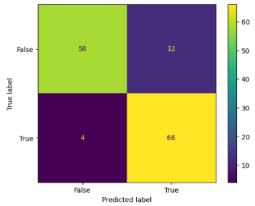


Inicialmente foi utilizado uma ELM com apenas 5 neurônios na camada intermediária. A matriz de confusão e o gráfico de separação mostram que a expansão aleatória com 5 neurônios está separando bem o modelo e não gerando overfitting.

```
[269]: seqx1x2 = np.linspace(start = -4, stop = 4, num = 300)
np_grid = seqx1x2.shape[0]
shape = (np_grid, np_grid)
MZ = np.zeros(shape)
for i in range(np_grid):
    for j in range(np_grid):
        x1 = seqx1x2[i]
        x2 = seqx1x2[j]
        x1x2 = np.column_stack((x1, x2, 1))
        h1 = np.tanh(np.dot(x1x2, Znormals))
        h1 = np.column_stack((h1, np.ones_like(h1[:, 0])))
        MZ[i, j] = np.sign(np.dot(h1, wnormals))[0]


plt.contour(seqx1x2, seqx1x2, MZ, levels = 1)
plt.scatter(x1_points[:, 0], x1_points[:, 1], color = 'red', label = 'data1')
plt.scatter(x2_points[:, 0], x2_points[:, 1], color = 'blue', label = 'data2')
plt.xlabel('x1')
plt.ylabel('x2')
plt.title('Contour Plot')
plt.xlim(-5, 5)
plt.ylim(-5, 5)
plt.grid(True)
plt.show()
```

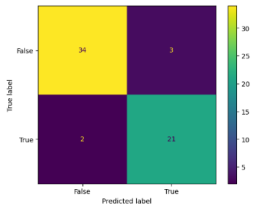



```
[265]: from sklearn import metrics
confusion_matrix = metrics.confusion_matrix(y_trainnormals, y_hatrainnormals)
cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix = confusion_matrix, display_labels = [False, True])
cm_display.plot()
plt.show()
sum_train = sum(y_trainnormals != y_hatrainnormals)
print(f"There are a total of {sum_train} errors in the train.")
```




```
There are a total of 16 errors in the train.
```

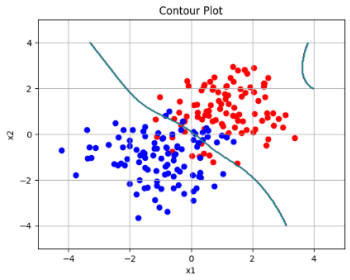
```
[266]: confusion_matrix = metrics.confusion_matrix(y_testnormals, y_hatestnormals)
cm_display_ = metrics.ConfusionMatrixDisplay(confusion_matrix = confusion_matrix, display_labels = [False, True])
cm_display_.plot()
plt.show()
sum_test = sum(y_testnormals != y_hatestnormals)
print(f"There are a total of {sum_test} errors in the test.")
```



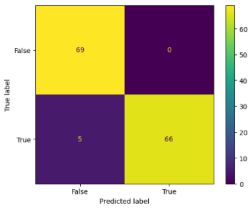

```
There are a total of 5 errors in the test.
```

Usando 10 neurônios na camada intermediária, podemos perceber uma tendência não linear de separação no gráfico dos dados.

```
[11]: seqx1x2 = np.linspace(start = -4, stop = 4, num = 300)
np_grid = seqx1x2.shape[0]
shape = (np_grid, np_grid)
MZ = np.zeros(shape)
for i in range(np_grid):
    for j in range(np_grid):
        x1 = seqx1x2[i]
        x2 = seqx1x2[j]
        x1x2 = np.column_stack((x1, x2, 1))
        h1 = np.tanh(np.dot(x1x2, Znormals))
        h1 = np.column_stack((h1, np.ones_like(h1[:, 0])))
        MZ[i, j] = np.sign(np.dot(h1, wnormals))[0]


plt.contour(seqx1x2, seqx1x2, MZ, levels = 1)
plt.scatter(x1_points[:, 0], x1_points[:, 1], color = 'red', label = 'data1')
plt.scatter(x2_points[:, 0], x2_points[:, 1], color = 'blue', label = 'data2')
plt.xlabel('x1')
plt.ylabel('x2')
plt.title('Contour Plot')
plt.xlim(-5, 5)
plt.ylim(-5, 5)
plt.grid(True)
plt.show()
```

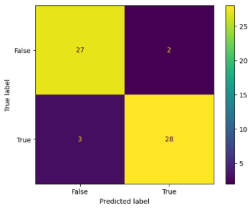



```
[276]: from sklearn import metrics
confusion_matrix = metrics.confusion_matrix(y_trainnormals, y_hatrainnormals)
cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix = confusion_matrix, display_labels = [False, True])
cm_display.plot()
plt.show()
sum_train = sum(y_trainnormals != y_hatrainnormals)
print(f"There are a total of {sum_train} errors in the train.")
```




```
There are a total of 5 errors in the train.
```

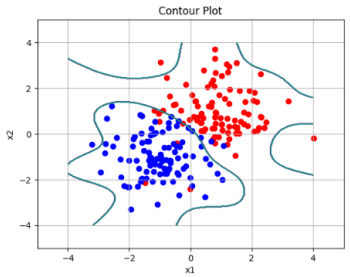
```
[277]: confusion_matrix = metrics.confusion_matrix(y_testnormals, y_hatestnormals)
cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix = confusion_matrix, display_labels = [False, True])
cm_display.plot()
plt.show()
sum_test = sum(y_testnormals != y_hatestnormals)
print(f"There are a total of {sum_test} errors in the test.")
```



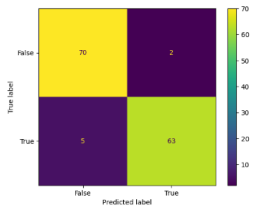

```
There are a total of 5 errors in the test.
```

Usando 30 neurônios na camada intermediária, é possível observar claramente uma boa separação dos dados, no gráfico de separação. No entanto, deve-se tomar cuidado para não colocar muitos neurônios na camada intermediária, gerar um modelo com overfitting e não conseguir generalizar uma boa saída para novos dados, diferentes dos de treinamento.

[291]:
```
seqx1x2 = np.linspace(start = -4, stop = 4, num = 300)
np_grid = seqx1x2.shape[0]
shape = (np_grid, np_grid)
MZ = np.zeros(shape)
for i in range(np_grid):
    for j in range(np_grid):
        x1 = seqx1x2[i]
        x2 = seqx1x2[j]
        x1x2 = np.column_stack((x1, x2, 1))
        h1 = np.tanh(np.dot(x1x2, Znormals))
        h1 = np.column_stack((h1, np.ones_like(h1[:, 0])))
        MZ[i, j] = np.sign(np.dot(h1, wnormals))[0]


plt.contour(seqx1x2, seqx1x2, MZ, levels = 1)
plt.scatter(x1_points[:, 0], x1_points[:, 1], color = 'red', label = 'data1')
plt.scatter(x2_points[:, 0], x2_points[:, 1], color = 'blue', label = 'data2')
plt.xlabel('x1')
plt.ylabel('x2')
plt.title('Contour Plot')
plt.xlim(-5, 5)
plt.ylim(-5, 5)
plt.grid(True)
plt.show()
```

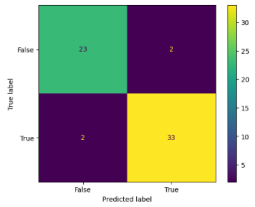


[287]:
```
from sklearn import metrics
confusion_matrix = metrics.confusion_matrix(y_trainnormals, y_hatrainnormals)
cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix = confusion_matrix, display_labels = [False, True])
cm_display.plot()
plt.show()
sum_train = sum(y_trainnormals != y_hatrainnormals)
print(f"There are a total of {sum_train} errors in the train.")
```




```
There are a total of 7 errors in the train.
```

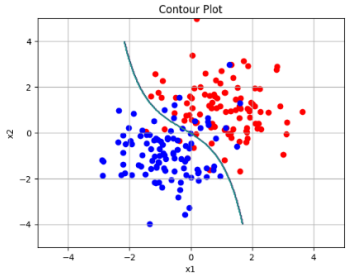
[288]:
```
confusion_matrix = metrics.confusion_matrix(y_testnormals, y_hatestnormals)
cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix = confusion_matrix, display_labels = [False, True])
cm_display.plot()
plt.show()
sum_test = sum(y_testnormals != y_hatestnormals)
print(f"There are a total of {sum_test} errors in the test.")
```




```
There are a total of 4 errors in the test.
```

Portanto, utilizando o modelo com apenas 6 neurônios na camada intermediária aparenta uma boa aproximação sem tendência a overfitting. A imagem abaixo mostra a separação utilizando 6 neuronios na camada intermediaria :

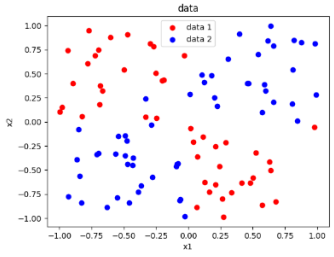
```
[44]: seqx1x2 = np.linspace(start = -4, stop = 4, num = 300)
      np_grid = seqx1x2.shape[0]
      shape = (np_grid, np_grid)
      MZ = np.zeros(shape)
      for i in range(np_grid):
          for j in range(np_grid):
              x1 = seqx1x2[i]
              x2 = seqx1x2[j]
              x1x2 = np.column_stack((x1, x2, 1))
              h1 = np.tanh(np.dot(x1x2, Znormals))
              h1 = np.column_stack((h1, np.ones_like(h1[:, 0])))
              MZ[i, j] = np.sign(np.dot(h1, wnormals))[0]


      plt.contour(seqx1x2, seqx1x2, MZ, levels = 1)
      plt.scatter(x1_points[:, 0], x1_points[:, 1], color = 'red', label = 'data1')
      plt.scatter(x2_points[:, 0], x2_points[:, 1], color = 'blue', label = 'data2')
      plt.xlabel('x1')
      plt.ylabel('x2')
      plt.title('Contour Plot')
      plt.xlim(-5, 5)
      plt.ylim(-5, 5)
      plt.grid(True)
      plt.show()
```



1.2 xor

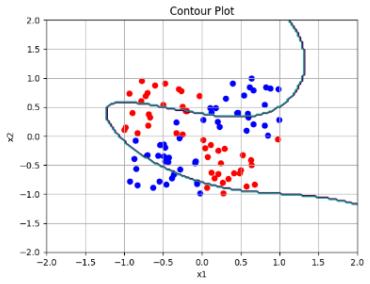
O segundo problema terá seus dados gerados também pelo pacote mlbench e são retratados abaixo :

```
[92]: x1_points = mat_plot[: 46, 0 : 2]
      x2_points = mat_plot[46 :, 0 : 2]
      plt.scatter(x1_points[:, 0], x1_points[:, 1], color = 'red', label = 'data 1')
      plt.scatter(x2_points[:, 0], x2_points[:, 1], color = 'blue', label = 'data 2')
      plt.title('data')
      plt.xlabel('x1')
      plt.ylabel('x2')
      plt.legend()
      plt.plot()

[92]: []
```

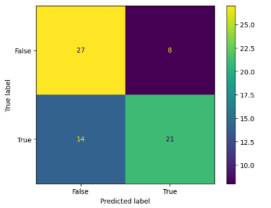


Inicialmente foi utilizado uma ELM com 5 neurônios na camada intermediária. Com apenas 5 neurônios na camada intermediária, é possível concluir que o modelo obteve um alto erro para os dados tanto de treino quanto de teste. A matriz de confusão e o gráfico de separação estão plotados abaixo :

```
[93]: seqx1x2 = np.linspace(start = -4, stop = 4, num = 300)
np_grid = seqx1x2.shape[0]
shape = (np_grid, np_grid)
MZ = np.zeros(shape)
for i in range(np_grid):
    for j in range(np_grid):
        x1 = seqx1x2[i]
        x2 = seqx1x2[j]
        x1x2 = np.column_stack((x1, x2, 1))
        h1 = np.tanh(np.dot(x1x2, Zxor))
        h1 = np.column_stack((h1, np.ones_like(h1[:, 0])))
        MZ[i, j] = np.sign(np.dot(h1, wxor))[0]


plt.contour(seqx1x2, seqx1x2, MZ, levels = 1)
plt.scatter(x1_points[:, 0], x1_points[:, 1], color = 'red', label = 'data1')
plt.scatter(x2_points[:, 0], x2_points[:, 1], color = 'blue', label = 'data2')
plt.xlabel('x1')
plt.ylabel('x2')
plt.title('Contour Plot')
plt.xlim(-2, 2)
plt.ylim(-2, 2)
plt.grid(True)
plt.show()
```

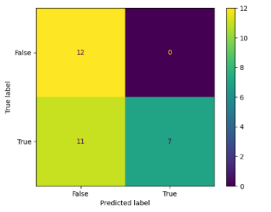



```
[88]: from sklearn import metrics
confusion_matrix = metrics.confusion_matrix(y_trainxor, y_hatrainxor)
cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix = confusion_matrix, display_labels = [False, True])
cm_display.plot()
plt.show()
sum_train = sum(y_trainxor != y_hatrainxor)
print(f"There are a total of {sum_train} errors in the train.")
```




```
There are a total of 22 errors in the train.
```

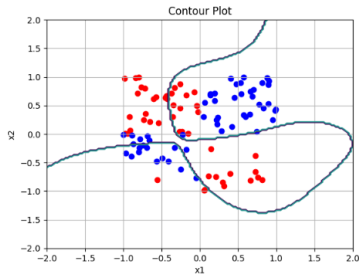
```
[89]: confusion_matrix = metrics.confusion_matrix(y_testxor, y_hatestxor)
cm_display_ = metrics.ConfusionMatrixDisplay(confusion_matrix = confusion_matrix, display_labels = [False, True])
cm_display_.plot()
plt.show()
sum_test = sum(y_testxor != y_hatestxor)
print(f"There are a total of {sum_test} errors in the test.")
```



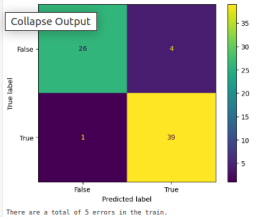

```
There are a total of 11 errors in the test.
```

Usando 10 neurônios na camada intermediária, podemos observar ainda uma boa separação e um menor erro para os dados de teste.

```
[105]: seqx1x2 = np.linspace(start = -4, stop = 4, num = 300)
np_grid = seqx1x2.shape[0]
shape = (np_grid, np_grid)
MZ = np.zeros(shape)
for i in range(np_grid):
    for j in range(np_grid):
        x1 = seqx1x2[i]
        x2 = seqx1x2[j]
        x1x2 = np.column_stack((x1, x2, 1))
        h1 = np.tanh(np.dot(x1x2, Zxor))
        h1 = np.column_stack((h1, np.ones_like(h1[:, 0])))
        MZ[i, j] = np.sign(np.dot(h1, wxor))[0]


plt.contour(seqx1x2, seqx1x2, MZ, levels = 1)
plt.scatter(x1_points[:, 0], x1_points[:, 1], color = 'red', label = 'data1')
plt.scatter(x2_points[:, 0], x2_points[:, 1], color = 'blue', label = 'data2')
plt.xlabel('x1')
plt.ylabel('x2')
plt.title('Contour Plot')
plt.xlim(-2, 2)
plt.ylim(-2, 2)
plt.grid(True)
plt.show()
```

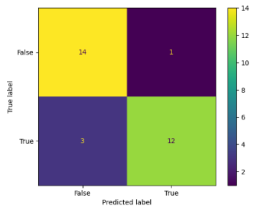



```
[100]: from sklearn import metrics
confusion_matrix = metrics.confusion_matrix(y_trainxor, y_hatrainxor)
cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix = confusion_matrix, display_labels = [False, True])
cm_display.plot()
plt.show()
sum_train = sum(y_trainxor != y_hatrainxor)
print(f"There are a total of {sum_train} errors in the train.")
```




```
There are a total of 5 errors in the train.
```

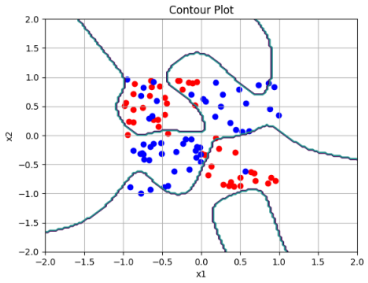
```
[101]: confusion_matrix = metrics.confusion_matrix(y_testxor, y_hatestxor)
cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix = confusion_matrix, display_labels = [False, True])
cm_display.plot()
plt.show()
sum_test = sum(y_testxor != y_hatestxor)
print(f"There are a total of {sum_test} errors in the test.")
```



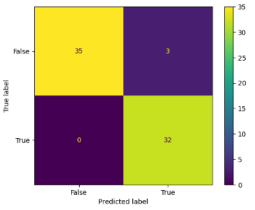

```
There are a total of 4 errors in the test.
```

Usando 30 neurônios na camada intermediária, é possível concluir que há um overfitting do modelo.

```
[117]: seqx1x2 = np.linspace(start = -4, stop = 4, num = 300)
np_grid = seqx1x2.shape[0]
shape = (np_grid, np_grid)
MZ = np.zeros(shape)
for i in range(np_grid):
    for j in range(np_grid):
        x1 = seqx1x2[i]
        x2 = seqx1x2[j]
        x1x2 = np.column_stack((x1, x2, 1))
        h1 = np.tanh(np.dot(x1x2, Zxor))
        h1 = np.column_stack((h1, np.ones_like(h1[:, 0])))
        MZ[i, j] = np.sign(np.dot(h1, wxor))[0]


plt.contour(seqx1x2, seqx1x2, MZ, levels = 1)
plt.scatter(x1_points[:, 0], x1_points[:, 1], color = 'red', label = 'data1')
plt.scatter(x2_points[:, 0], x2_points[:, 1], color = 'blue', label = 'data2')
plt.xlabel('x1')
plt.ylabel('x2')
plt.title('Contour Plot')
plt.xlim(-2, 2)
plt.ylim(-2, 2)
plt.grid(True)
plt.show()
```

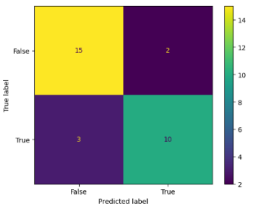



```
[112]: from sklearn import metrics
confusion_matrix = metrics.confusion_matrix(y_trainxor, y_hatrainxor)
cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix = confusion_matrix, display_labels = [False, True])
cm_display.plot()
plt.show()
sum_train = sum(y_trainxor != y_hatrainxor)
print(f"There are a total of {sum_train} errors in the train.")
```




```
There are a total of 3 errors in the train.
```

```
[113]: confusion_matrix = metrics.confusion_matrix(y_testxor, y_hatestxor)
cm_display_ = metrics.ConfusionMatrixDisplay(confusion_matrix = confusion_matrix, display_labels = [False, True])
cm_display_.plot()
plt.show()
sum_test = sum(y_testxor != y_hatestxor)
print(f"There are a total of {sum_test} errors in the test.")
```



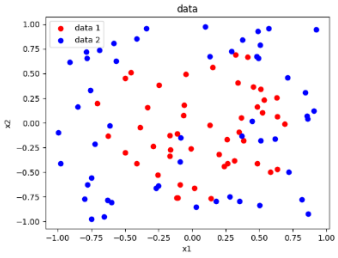

```
There are a total of 5 errors in the test.
```

Portanto, podemos concluir que o modelo com 10 neuronios na camada intermediaria gerará uma boa saída para os dados de entrada.

1.3 circle

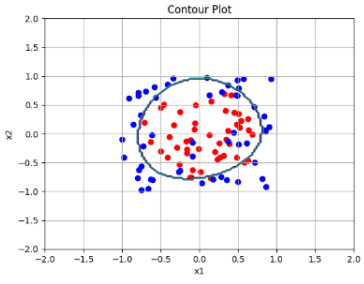
O terceiro problema terá seus dados gerados também pelo pacote mlbench e são retratados abaixo :

```
[85]: x1_points = mat_plot[: 48, 0 : 2]
      x2_points = mat_plot[48 :, 0 : 2]
      plt.scatter(x1_points[:, 0], x1_points[:, 1], color = 'red', label = 'data 1')
      plt.scatter(x2_points[:, 0], x2_points[:, 1], color = 'blue', label = 'data 2')
      plt.title('data')
      plt.xlabel('x1')
      plt.ylabel('x2')
      plt.legend()
      plt.plot()

[85]: []
```

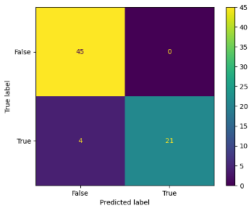


Com apenas 5 neurônios na camada intermediária, é possível concluir que o modelo obteve um alto erro para os dados tanto de treino quanto de teste. A matriz de confusão e o gráfico de separação estão plotados abaixo :

```
[86]: seqx1x2 = np.linspace(start = -4, stop = 4, num = 300)
      np_grid = seqx1x2.shape[0]
      shape = (np_grid, np_grid)
      MZ = np.zeros(shape)
      for i in range(np_grid):
          for j in range(np_grid):
              x1 = seqx1x2[i]
              x2 = seqx1x2[j]
              x1x2 = np.column_stack((x1, x2, 1))
              h1 = np.tanh(np.dot(x1x2, Zcircle))
              h1 = np.column_stack((h1, np.ones_like(h1[:, 0])))
              MZ[i, j] = np.sign(np.dot(h1, wcircle))[0]


      plt.contour(seqx1x2, seqx1x2, MZ, levels = 1)
      plt.scatter(x1_points[:, 0], x1_points[:, 1], color = 'red', label = 'data1')
      plt.scatter(x2_points[:, 0], x2_points[:, 1], color = 'blue', label = 'data2')
      plt.xlabel('x1')
      plt.ylabel('x2')
      plt.title('Contour Plot')
      plt.xlim(-2, 2)
      plt.ylim(-2, 2)
      plt.grid(True)
      plt.show()
```

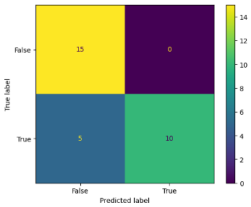


```
[81]: from sklearn import metrics
      confusion_matrix = metrics.confusion_matrix(y_traincircle, y_hatraincircle)
      cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix = confusion_matrix, display_labels = [False, True])
      cm_display.plot()
      plt.show()
      sum_train = sum(y_traincircle != y_hatraincircle)
      print(f"There are a total of {sum_train} errors in the train.")
```




```
There are a total of 4 errors in the train.
```

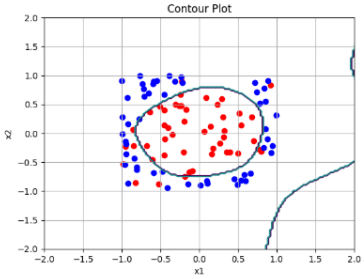
```
[82]: confusion_matrix = metrics.confusion_matrix(y_testcircle, y_hatestcircle)
      cm_display_ = metrics.ConfusionMatrixDisplay(confusion_matrix = confusion_matrix, display_labels = [False, True])
      cm_display_.plot()
      plt.show()
      sum_test = sum(y_testcircle != y_hatestcircle)
      print(f"There are a total of {sum_test} errors in the test.")
```

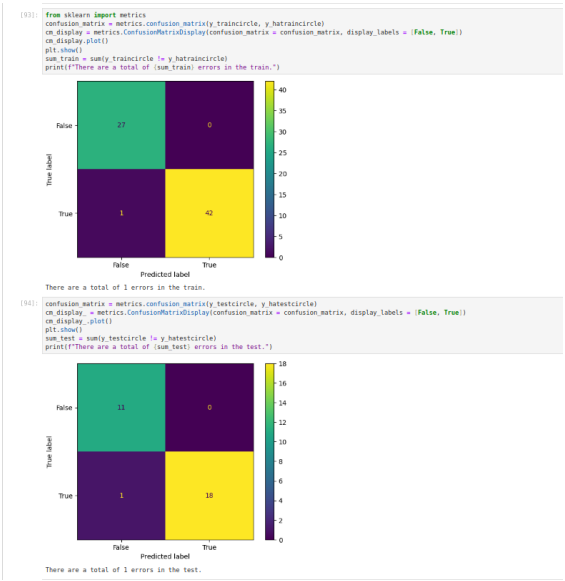



```
There are a total of 5 errors in the test.
```

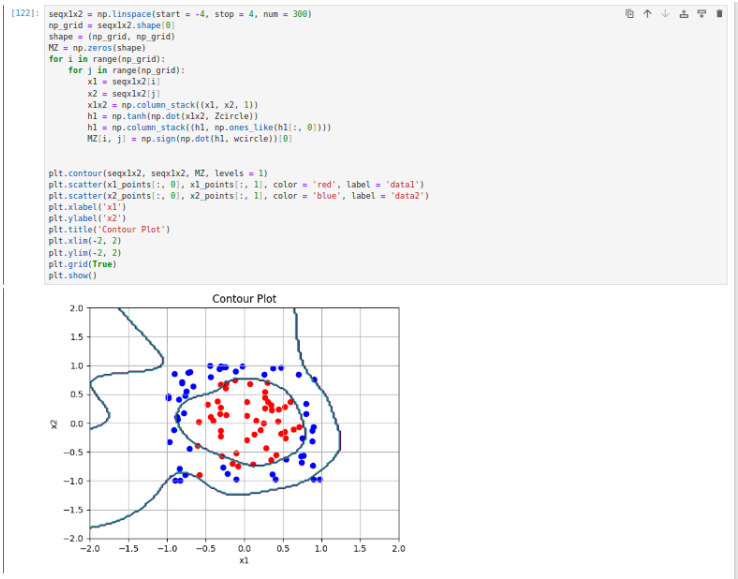
Usando 10 neurônios na camada intermediária, podemos observar ainda uma boa separação e um menor erro para os dados de teste.

```
[98]: seqx1x2 = np.linspace(start = -4, stop = 4, num = 300)
      np_grid = seqx1x2.shape[0]
      shape = (np_grid, np_grid)
      MZ = np.zeros(shape)
      for i in range(np_grid):
          for j in range(np_grid):
              x1 = seqx1x2[i]
              x2 = seqx1x2[j]
              x1x2 = np.column_stack((x1, x2, 1))
              h1 = np.tanh(np.dot(x1x2, Zcircle))
              h1 = np.column_stack((h1, np.ones_like(h1[:, 0])))
              MZ[i, j] = np.sign(np.dot(h1, wcircle))[0]


      plt.contour(seqx1x2, seqx1x2, MZ, levels = 1)
      plt.scatter(x1_points[:, 0], x1_points[:, 1], color = 'red', label = 'data1')
      plt.scatter(x2_points[:, 0], x2_points[:, 1], color = 'blue', label = 'data2')
      plt.xlabel('x1')
      plt.ylabel('x2')
      plt.title('Contour Plot')
      plt.xlim(-2, 2)
      plt.ylim(-2, 2)
      plt.grid(True)
      plt.show()
```

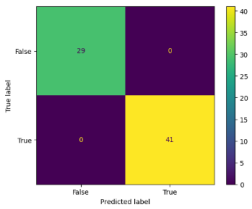




Usando 30 neurônios na camada intermediária, é possível concluir que há um overfitting do modelo.

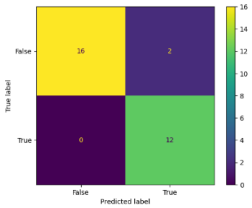


```
[117]: from sklearn import metrics
confusion_matrix = metrics.confusion_matrix(y_traincircle, y_hatraincircle)
cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix = confusion_matrix, display_labels = [False, True])
cm_display.plot()
plt.show()
sum_train = sum(y_traincircle != y_hatraincircle)
print(f"There are a total of {sum_train} errors in the train.")
```




```
There are a total of 0 errors in the train.
```

```
[118]: confusion_matrix = metrics.confusion_matrix(y_testcircle, y_hatestcircle)
cm_display_ = metrics.ConfusionMatrixDisplay(confusion_matrix = confusion_matrix, display_labels = [False, True])
cm_display_.plot()
plt.show()
sum_test = sum(y_testcircle != y_hatestcircle)
print(f"There are a total of {sum_test} errors in the test.")
```




```
There are a total of 2 errors in the test.
```

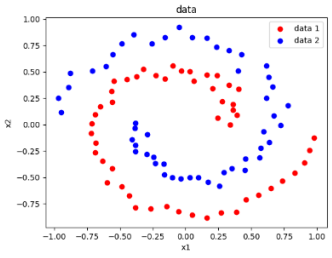
Portanto, podemos concluir que o modelo com 10 neuronios na camada intermediaria gerará uma boa saída para os dados de entrada.

1.4 spirals

O último problema terá seus dados gerados também pelo pacote mlbench, contém um formato espiral e são retratados abaixo :

```
[71]: x1_points = mat_plot[: 50, 0 : 2]
x2_points = mat_plot[50 :, 0 : 2]
plt.scatter(x1_points[:, 0], x1_points[:, 1], color = 'red', label = 'data 1')
plt.scatter(x2_points[:, 0], x2_points[:, 1], color = 'blue', label = 'data 2')
plt.title('data')
plt.xlabel('x1')
plt.ylabel('x2')
plt.legend()
plt.plot()
```

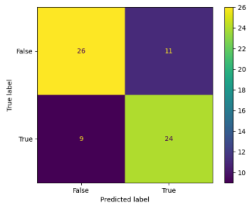
```
[71]: []
```



Inicialmente foi utilizado uma ELM com 5 neurônios na camada intermediária. Com apenas 5 neurônios na camada intermediária, é possível concluir que o modelo obteve um alto erro para os dados tanto de treino quanto de teste, os erros foram bastante altos e podemos considerar isso como um underfitting. A matriz de confusão e o gráfico de separação estão plotados abaixo :

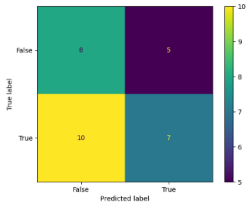
```
[72]: seqx1x2 = np.linspace(start = -4, stop = 4, num = 300)
np_grid = seqx1x2.shape[0]
shape = (np_grid, np_grid)
MZ = np.zeros(shape)
for i in range(np_grid):
    for j in range(np_grid):
        x1 = seqx1x2[i]
        x2 = seqx1x2[j]
        x1x2 = np.column_stack((x1, x2, 1))
        h1 = np.tanh(np.dot(x1x2, Zspirals))
        h1 = np.column_stack((h1, np.ones_like(h1[:, 0])))
        MZ[i, j] = np.sign(np.dot(h1, wspirals))[0]


plt.contour(seqx1x2, seqx1x2, MZ, levels = 1)
plt.scatter(x1_points[:, 0], x1_points[:, 1], color = 'red', label = 'data1')
plt.scatter(x2_points[:, 0], x2_points[:, 1], color = 'blue', label = 'data2')
plt.xlabel('x1')
plt.ylabel('x2')
plt.title('Contour Plot')
plt.xlim(-2, 2)
plt.ylim(-2, 2)
plt.grid(True)
plt.show()
```


```
[67]: from sklearn import metrics
confusion_matrix = metrics.confusion_matrix(y_trainspirals, y_hatrainspirals)
cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix = confusion_matrix, display_labels = [False, True])
cm_display.plot()
plt.show()
sum_train = sum(y_trainspirals != y_hatrainspirals)
print(f"There are a total of {sum_train} errors in the train.")
```




```
There are a total of 20 errors in the train.
```

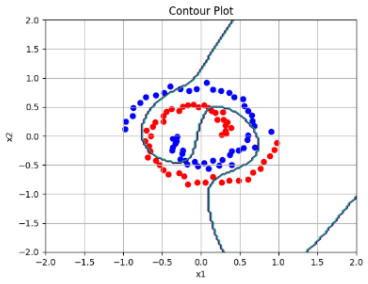
```
[68]: confusion_matrix = metrics.confusion_matrix(y_testspirals, y_hatestspirals)
cm_display_ = metrics.ConfusionMatrixDisplay(confusion_matrix = confusion_matrix, display_labels = [False, True])
cm_display_.plot()
plt.show()
sum_test = sum(y_testspirals != y_hatestspirals)
print(f"There are a total of {sum_test} errors in the test.")
```



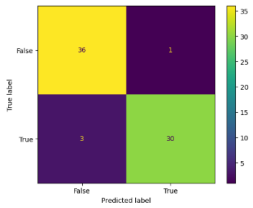

```
There are a total of 15 errors in the test.
```

Usando 10 neurônios na camada intermediária, podemos observar ainda uma boa separação e um menor erro para os dados de teste.

```
[84]: seqx1x2 = np.linspace(start = -4, stop = 4, num = 300)
np_grid = seqx1x2.shape[0]
shape = (np_grid, np_grid)
MZ = np.zeros(shape)
for i in range(np_grid):
    for j in range(np_grid):
        x1 = seqx1x2[i]
        x2 = seqx1x2[j]
        x1x2 = np.column_stack((x1, x2, 1))
        h1 = np.tanh(np.dot(x1x2, Zspirals))
        h1 = np.column_stack((h1, np.ones_like(h1[:, 0])))
        MZ[i, j] = np.sign(np.dot(h1, wspirals))[0]


plt.contour(seqx1x2, seqx1x2, MZ, levels = 1)
plt.scatter(x1_points[:, 0], x1_points[:, 1], color = 'red', label = 'data1')
plt.scatter(x2_points[:, 0], x2_points[:, 1], color = 'blue', label = 'data2')
plt.xlabel('x1')
plt.ylabel('x2')
plt.title('Contour Plot')
plt.xlim(-2, 2)
plt.ylim(-2, 2)
plt.grid(True)
plt.show()
```

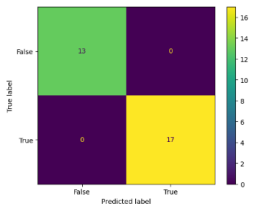



```
[79]: from sklearn import metrics
confusion_matrix = metrics.confusion_matrix(y_trainspirals, y_hatrainspirals)
cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix = confusion_matrix, display_labels = [False, True])
cm_display.plot()
plt.show()
sum_train = sum(y_trainspirals != y_hatrainspirals)
print(f"There are a total of {sum_train} errors in the train.")
```



There are a total of 4 errors in the train.

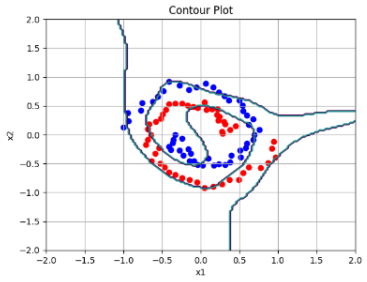
```
[80]: confusion_matrix = metrics.confusion_matrix(y_testspirals, y_hatestspirals)
cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix = confusion_matrix, display_labels = [False, True])
cm_display.plot()
plt.show()
sum_test = sum(y_testspirals != y_hatestspirals)
print(f"There are a total of {sum_test} errors in the test.")
```



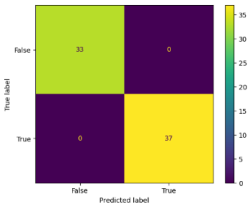
There are a total of 0 errors in the test.

Usando 30 neurônios na camada intermediária, é possível concluir que há um overfitting do modelo.

```
[96]: seqx1x2 = np.linspace(start = -4, stop = 4, num = 300)
np_grid = seqx1x2.shape[0]
shape = (np_grid, np_grid)
MZ = np.zeros(shape)
for i in range(np_grid):
    for j in range(np_grid):
        x1 = seqx1x2[i]
        x2 = seqx1x2[j]
        x1x2 = np.column_stack((x1, x2, 1))
        h1 = np.tanh(np.dot(x1x2, Zspirals))
        h1 = np.column_stack((h1, np.ones_like(h1[:, 0])))
        MZ[i, j] = np.sign(np.dot(h1, wspirals))[0]


plt.contour(seqx1x2, seqx1x2, MZ, levels = 1)
plt.scatter(x1_points[:, 0], x1_points[:, 1], color = 'red', label = 'data1')
plt.scatter(x2_points[:, 0], x2_points[:, 1], color = 'blue', label = 'data2')
plt.xlabel('x1')
plt.ylabel('x2')
plt.title('Contour Plot')
plt.xlim(-2, 2)
plt.ylim(-2, 2)
plt.grid(True)
plt.show()
```

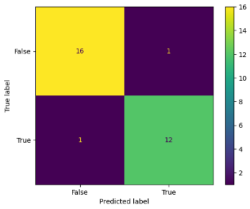



```
[91]: from sklearn import metrics
confusion_matrix = metrics.confusion_matrix(y_trainspirals, y_hatrainspirals)
cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix = confusion_matrix, display_labels = [False, True])
cm_display.plot()
plt.show()
sum_train = sum(y_trainspirals != y_hatrainspirals)
print(f"There are a total of {sum_train} errors in the train.")
```




```
There are a total of 0 errors in the train.
```

```
[92]: confusion_matrix = metrics.confusion_matrix(y_testspirals, y_hatestspirals)
cm_display_ = metrics.ConfusionMatrixDisplay(confusion_matrix = confusion_matrix, display_labels = [False, True])
cm_display_.plot()
plt.show()
sum_test = sum(y_testspirals != y_hatestspirals)
print(f"There are a total of {sum_test} errors in the test.")
```




```
There are a total of 2 errors in the test.
```

Portanto, podemos concluir que o modelo com 10 neuronios na camada intermediaria gerará uma boa saída para os dados de entrada. Para observar o resultado, basta analisar a matriz de confusão dos dados de teste ao usar 10 neuronios na camada intermediária. O erro sobre os dados de teste foi 0 e portanto o modelo generalizou bem a função geradora.