Exercício 5

Arthur Felipe Reis Souza

April 14, 2024

1 Introduction

O exercício consiste em aplicar as rede neurais do tipo ELM para resolver problemas multidimensionais. Serão utilizados 2 conjunto de dados, o conjunto de dados Breast Cancer e o conjunto de dados Statlog.

1.1 Breast Cancer

O dataset Breast Cancer foi utilizado.

192]:

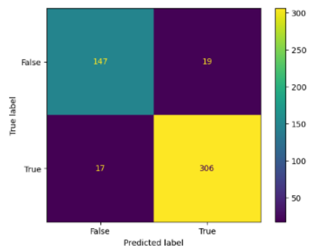
	Id	Cl.thickness	Cell.size	Cell.shape	Marg.adhesion	Epith.c.size	Bare.nuclei	Bl.cromatin	Normal.nucleoli	Mitoses	Class
1	1000025	5	1	1	1	2	1	3	1	1	1
2	1002945	5	4	4	5	7	10	3	2	1	1
3	1015425	3	1	1	1	2	2	3	1	1	1
4	1016277	6	8	8	1	3	4	3	7	1	1
5	1017023	4	1	1	3	2	1	3	1	1	1
...	...	...	...	...	...	...	...	...	...	...	...
695	776715	3	1	1	1	3	2	1	1	1	1
696	841769	2	1	1	1	2	1	1	1	1	1
697	888820	5	10	10	3	7	3	8	10	2	-1
698	897471	4	8	6	4	3	4	10	6	1	-1
699	897471	4	8	8	5	4	5	10	4	1	-1

699 rows × 11 columns

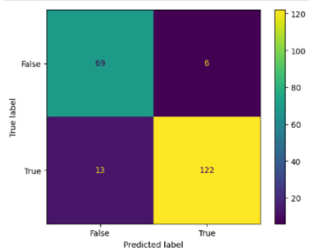
193]:

```
ion import train_test_split
 y_test = train_test_split(Breast_Cancer_PANDAS_df.iloc[:, 1 : 10], Breast_Cancer_PANDAS_df.iloc[:, 10], random_state = 0, train_size
```

Inicialmente foi utilizado uma rede 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 :



[120]:
```
from sklearn import metrics
confusion_matrix = metrics.confusion_matrix(y_test, y_hat_test)
cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix = confusion_matrix, display_labels = [False, True])
cm_display.plot()
plt.show()
```



[138]:
```
import train_test_ELM
from sklearn.metrics import accuracy_score
p = 5
X_train = np.array(X_train)
X_test = np.array(X_test)
y_train = np.array(y_train)
y_test = np.array(y_test)
mean_acc = 0
lst_of_results = list()
for i in range(10):
    train_ELM = train_test_ELM.train_ELM(X_train, y_train, p, control = True)
    w = np.array(train_ELM[0])
    H = np.array(train_ELM[1])
    Z = np.array(train_ELM[2])
    y_hat_test = train_test_ELM.test_ELM(X_test, Z, w, True)
    lst_of_results.append(accuracy_score(y_test, y_hat_test))
    mean_acc += accuracy_score(y_test, y_hat_test)
mean_acc = (mean_acc / 10)
lst_of_results = np.array(lst_of_results)
stand_dev = 0
for i in range(10):
    stand_dev = (lst_of_results[i] - mean_acc) ** 2
stand_dev = stand_dev / lst_of_results.shape[0]
```

[139]:
```
print(f'A acurácia média para 10 amostras é : {mean_acc * 100}% +- {stand_dev}')
```

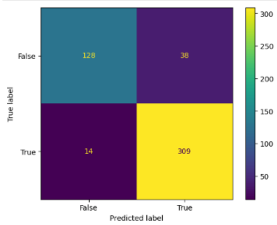
```
A acurácia média para 10 amostras é : 85.28571428571429% +- 3.448979591836735e-05
```

[140]:
```
y_hat_train = train_test_ELM.test_ELM(X_train, Z, w, True)
```

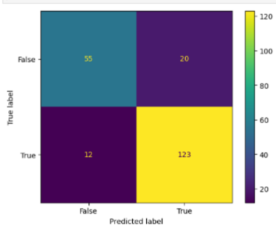
[141]:
```
from sklearn import metrics
confusion_matrix = metrics.confusion_matrix(y_train, y_hat_train)
cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix = confusion_matrix, display_labels = [False, True])
cm_display.plot()
plt.show()
```

Utilizando com 10 neuronios obtemos os resultados :

```
[152]: from sklearn import metrics
confusion_matrix = metrics.confusion_matrix(y_train, y_hat_train)
cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix = confusion_matrix, display_labels = [False, True])
cm_display.plot()
plt.show()
```




```
[153]: from sklearn import metrics
confusion_matrix = metrics.confusion_matrix(y_test, y_hat_test)
cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix = confusion_matrix, display_labels = [False, True])
cm_display.plot()
plt.show()
```




```
[149]: import train_test_ELM
from sklearn.metrics import accuracy_score
p = 10
X_train = np.array(X_train)
X_test = np.array(X_test)
y_train = np.array(y_train)
y_test = np.array(y_test)
mean_acc = 0
lst_of_results = list()
for i in range(10):
    train_ELM = train_test_ELM.train_ELM(X_train, y_train, p, control = True)
    w = np.array(train_ELM[0])
    H = np.array(train_ELM[1])
    Z = np.array(train_ELM[2])
    y_hat_test = train_test_ELM.test_ELM(X_test, Z, w, True)
    lst_of_results.append(accuracy_score(y_test, y_hat_test))
    mean_acc += accuracy_score(y_test, y_hat_test)
mean_acc = (mean_acc / 10)
lst_of_results = np.array(lst_of_results)
stand_dev = 0
for i in range(10):
    stand_dev = (lst_of_results[i] - mean_acc) ** 2
stand_dev = stand_dev / lst_of_results.shape[0]
```

```
[150]: print(f'A acurácia média para 10 amostras é : {mean_acc * 100}% +- {stand_dev}')
```

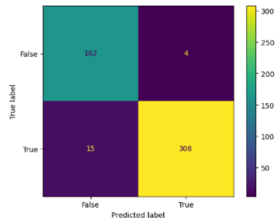
```
A acurácia média para 10 amostras é : 87.71428571428575% +- 8.716553287982047e-05
```

```
[151]: y_hat_train = train_test_ELM.test_ELM(X_train, Z, w, True)
```

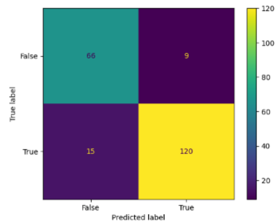
```
[152]: from sklearn import metrics
confusion_matrix = metrics.confusion_matrix(y_train, y_hat_train)
cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix = confusion_matrix, display_labels = [False, True])
cm_display.plot()
plt.show()
```

Utilizando com 30 neuronios obtemos os resultados :

```
[163]: from sklearn import metrics
confusion_matrix = metrics.confusion_matrix(y_train, y_hat_train)
cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix = confusion_matrix, display_labels = [False, True])
cm_display.plot()
plt.show()
```




```
[164]: from sklearn import metrics
confusion_matrix = metrics.confusion_matrix(y_test, y_hat_test)
cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix = confusion_matrix, display_labels = [False, True])
cm_display.plot()
plt.show()
```

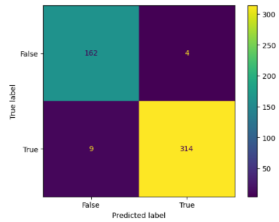



```
[160]: import train_test_ELM
from sklearn.metrics import accuracy_score
p = 30
X_train = np.array(X_train)
X_test = np.array(X_test)
y_train = np.array(y_train)
y_test = np.array(y_test)
mean_acc = 0
lst_of_results = list()
for i in range(10):
    train_ELM = train_test_ELM.train_ELM(X_train, y_train, p, control = True)
    w = np.array(train_ELM[0])
    H = np.array(train_ELM[1])
    Z = np.array(train_ELM[2])
    y_hat_test = train_test_ELM.test_ELM(X_test, Z, w, True)
    lst_of_results.append(accuracy_score(y_test, y_hat_test))
    mean_acc += accuracy_score(y_test, y_hat_test)
mean_acc = (mean_acc / 10)
lst_of_results = np.array(lst_of_results)
stand_dev = 0
for i in range(10):
    stand_dev = (lst_of_results[i] - mean_acc) ** 2
stand_dev = stand_dev / lst_of_results.shape[0]
```

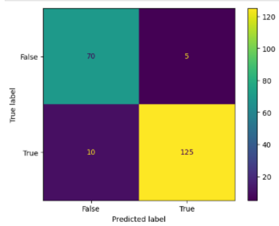
```
[161]: print(f'A acurácia média para 10 amostras é : {mean_acc * 100}% +- {stand_dev}')
A acurácia média para 10 amostras é : 92.04761904761905% +- 0.00012083900226757453
```

Utilizando com 50 neuronios obtemos os resultados :

```
[174]: from sklearn import metrics
confusion_matrix = metrics.confusion_matrix(y_train, y_hat_train)
cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix = confusion_matrix, display_labels = [False, True])
cm_display.plot()
plt.show()
```




```
[175]: from sklearn import metrics
confusion_matrix = metrics.confusion_matrix(y_test, y_hat_test)
cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix = confusion_matrix, display_labels = [False, True])
cm_display.plot()
plt.show()
```



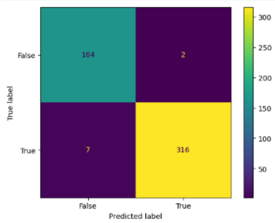

```
[171]: import train_test_ELM
from sklearn.metrics import accuracy_score
p = 50
X_train = np.array(X_train)
X_test = np.array(X_test)
y_train = np.array(y_train)
y_test = np.array(y_test)
mean_acc = 0
lst_of_results = list()
for i in range(10):
    train_ELM = train_test_ELM.train_ELM(X_train, y_train, p, control = True)
    w = np.array(train_ELM[0])
    H = np.array(train_ELM[1])
    Z = np.array(train_ELM[2])
    y_hat_test = train_test_ELM.test_ELM(X_test, Z, w, True)
    lst_of_results.append(accuracy_score(y_test, y_hat_test))
    mean_acc += accuracy_score(y_test, y_hat_test)
mean_acc = (mean_acc / 10)
lst_of_results = np.array(lst_of_results)
stand_dev = 0
for i in range(10):
    stand_dev = (lst_of_results[i] - mean_acc) ** 2
stand_dev = stand_dev / lst_of_results.shape[0]
```

```
[172]: print(f'A acurácia média para 10 amostras é : {mean_acc * 100}% +- {stand_dev}')
A acurácia média para 10 amostras é : 92.71428571428572% +- 2.040816326530344e-07
```

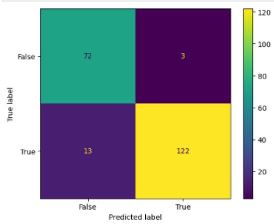
```
[173]: y_hat_train = train_test_ELM.test_ELM(X_train, Z, w, True)
```

Utilizando com 100 neuronios obtemos os resultados :

[186]:
```python
from sklearn import metrics
confusion_matrix = metrics.confusion_matrix(y_train, y_hat_train)
cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix = confusion_matrix, display_labels = [False, True])
cm_display.plot()
plt.show()
```



[187]:
```python
from sklearn import metrics
confusion_matrix = metrics.confusion_matrix(y_test, y_hat_test)
cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix = confusion_matrix, display_labels = [False, True])
cm_display.plot()
plt.show()
```



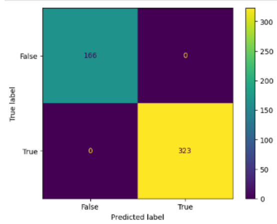
[183]:
```python
import train_test_ELM
from sklearn.metrics import accuracy_score
p = 100
X_train = np.array(X_train)
X_test = np.array(X_test)
y_train = np.array(y_train)
y_test = np.array(y_test)
mean_acc = 0
lst_of_results = list()
for i in range(10):
    train_ELM = train_test_ELM.train_ELM(X_train, y_train, p, control = True)
    w = np.array(train_ELM[0])
    H = np.array(train_ELM[1])
    Z = np.array(train_ELM[2])
    y_hat_test = train_test_ELM.test_ELM(X_test, Z, w, True)
    lst_of_results.append(accuracy_score(y_test, y_hat_test))
    mean_acc += accuracy_score(y_test, y_hat_test)
mean_acc = (mean_acc / 10)
lst_of_results = np.array(lst_of_results)
stand_dev = 0
for i in range(10):
    stand_dev = (lst_of_results[i] - mean_acc) ** 2
stand_dev = stand_dev / lst_of_results.shape[0]
```

[184]:
```python
print(f'A acurácia média para 10 amostras é : {mean_acc * 100}% +- {stand_dev}')
```

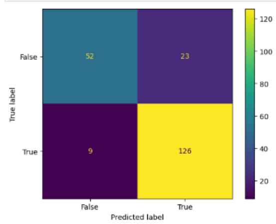
```
A acurácia média para 10 amostras é : 92.28571428571429% +- 9.070294784580011e-08
```

Utilizando com 300 neuronios obtemos os resultados :

```
[197]: from sklearn import metrics
       confusion_matrix = metrics.confusion_matrix(y_train, y_hat_train)
       cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix = confusion_matrix, display_labels = [False, True])
       cm_display.plot()
       plt.show()
```




```
[198]: from sklearn import metrics
       confusion_matrix = metrics.confusion_matrix(y_test, y_hat_test)
       cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix = confusion_matrix, display_labels = [False, True])
       cm_display.plot()
       plt.show()
```

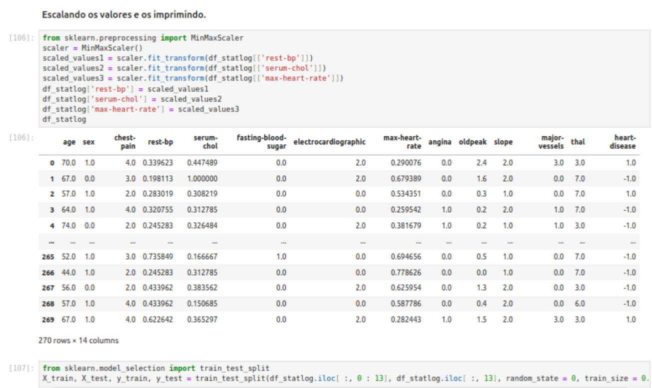



```
[194]: import train_test_ELM
       from sklearn.metrics import accuracy_score
       p = 300
       X_train = np.array(X_train)
       X_test = np.array(X_test)
       y_train = np.array(y_train)
       y_test = np.array(y_test)
       mean_acc = 0
       lst_of_results = list()
       for i in range(10):
           train_ELM = train_test_ELM.train_ELM(X_train, y_train, p, control = True)
           w = np.array(train_ELM[0])
           H = np.array(train_ELM[1])
           Z = np.array(train_ELM[2])
           y_hat_test = train_test_ELM.test_ELM(X_test, Z, w, True)
           lst_of_results.append(accuracy_score(y_test, y_hat_test))
           mean_acc += accuracy_score(y_test, y_hat_test)
       mean_acc = (mean_acc / 10)
       lst_of_results = np.array(lst_of_results)
       stand_dev = 0
       for i in range(10):
           stand_dev = (lst_of_results[i] - mean_acc) ** 2
       stand_dev = stand_dev / lst_of_results.shape[0]

[195]: print(f'A acurácia média para 10 amostras é : {mean_acc * 100}% +- {stand_dev}')

       A acurácia média para 10 amostras é : 81.57142857142856% +- 0.0001017913832199552
```

Portanto, observando as matrizes de confusão e as acurácias médias, é possível afirmar que o modelo, ao usar 300 neuronios na camada intermediária é 1 modelo com overfitting. A acurácia permanece a mesma com 30, 50 e 100 neurônios. Para reduzir o custo computacional, considerando-se que eles contém uma mesma acurácia, o modelo com 30 neuronios na camada intermediária generaliza melhor.

1.2 Hearth Disease

O dataset Hearth Disease foi utilizado.



Escalando os valores e os imprimindo.

[106]:
```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
scaled_values1 = scaler.fit_transform(df_statlog[['rest-bp']])
scaled_values2 = scaler.fit_transform(df_statlog[['serum-chol']])
scaled_values3 = scaler.fit_transform(df_statlog[['max-heart-rate']])
df_statlog['rest-bp'] = scaled_values1
df_statlog['serum-chol'] = scaled_values2
df_statlog['max-heart-rate'] = scaled_values3
df_statlog
```

[106]:

	age	sex	chest-pain	rest-bp	serum-chol	fasting-blood-sugar	electrocardiographic	max-heart-rate	angina	oldpeak	slope	major-vessels	thal	heart-disease
0	70.0	1.0	4.0	0.339623	0.447489	0.0	2.0	0.290076	0.0	2.4	2.0	3.0	3.0	1.0
1	67.0	0.0	3.0	0.198113	1.000000	0.0	2.0	0.679389	0.0	1.6	2.0	0.0	7.0	-1.0
2	57.0	1.0	2.0	0.283019	0.308219	0.0	0.0	0.534351	0.0	0.3	1.0	0.0	7.0	1.0
3	64.0	1.0	4.0	0.320755	0.312785	0.0	0.0	0.259542	1.0	0.2	2.0	1.0	7.0	-1.0
4	74.0	0.0	2.0	0.245283	0.326484	0.0	2.0	0.381679	1.0	0.2	1.0	1.0	3.0	-1.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
265	52.0	1.0	3.0	0.735849	0.166667	1.0	0.0	0.694656	0.0	0.5	1.0	0.0	7.0	-1.0
266	44.0	1.0	2.0	0.245283	0.312785	0.0	0.0	0.778626	0.0	0.0	1.0	0.0	7.0	-1.0
267	56.0	0.0	2.0	0.433962	0.383562	0.0	2.0	0.625954	0.0	1.3	2.0	0.0	3.0	-1.0
268	57.0	1.0	4.0	0.433962	0.150685	0.0	0.0	0.587786	0.0	0.4	2.0	0.0	6.0	-1.0
269	67.0	1.0	4.0	0.622642	0.365297	0.0	2.0	0.282443	1.0	1.5	2.0	3.0	3.0	1.0

270 rows × 14 columns

[107]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(df_statlog.iloc[ :, 0 : 13], df_statlog.iloc[ :, 13], random_state = 0, train_size = 0.
```

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 :

```
[128]: import train_test_ELM
from sklearn.metrics import accuracy_score
p = 5
X_train = np.array(X_train)
X_test = np.array(X_test)
y_train = np.array(y_train)
y_test = np.array(y_test)
mean_acc = 0
lst_of_results = list()
for i in range(10):
    train_ELM = train_test_ELM.train_ELM(X_train, y_train, p, control = True)
    w = np.array(train_ELM[0])
    H = np.array(train_ELM[1])
    Z = np.array(train_ELM[2])
    y_hatest = train_test_ELM.test_ELM(X_test, Z, w, True)
    mean_acc += accuracy_score(y_test, y_hatest)
    lst_of_results.append(accuracy_score(y_test, y_hatest))
mean_acc = (mean_acc / 10)
stand_dev = 0
lst_of_results = np.array(lst_of_results)
for i in range(10):
    stand_dev = (lst_of_results[i] - mean_acc) ** 2
stand_dev = stand_dev / lst_of_results.shape[0]
```

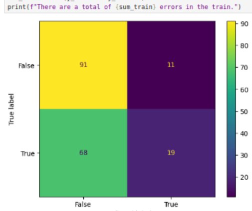
```
[129]: print(f'A acurácia média para 10 amostras é : {mean_acc * 100}% +- {stand_dev}')
```

```
A acurácia média para 10 amostras é : 63.33333333333333% +- 8.062795305593631e-05
```

```
[130]: y_hatrain = train_test_ELM.test_ELM(X_train, Z, w, True)
```

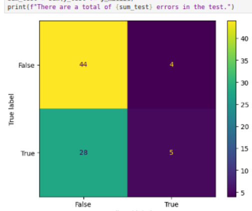
Utilizando com 10 neuronios obtemos os resultados :

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




```
There are a total of 79 errors in the train.
```

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




```
There are a total of 32 errors in the test.
```

```
[138]: import train_test_ELM
from sklearn.metrics import accuracy_score
p = 10
X_train = np.array(X_train)
X_test = np.array(X_test)
y_train = np.array(y_train)
y_test = np.array(y_test)
mean_acc = 0
lst_of_results = list()
for i in range(10):
    train_ELM = train_test_ELM.train_ELM(X_train, y_train, p, control = True)
    w = np.array(train_ELM[0])
    H = np.array(train_ELM[1])
    Z = np.array(train_ELM[2])
    y_hatest = train_test_ELM.test_ELM(X_test, Z, w, True)
    mean_acc += accuracy_score(y_test, y_hatest)
    lst_of_results.append(accuracy_score(y_test, y_hatest))
mean_acc = (mean_acc / 10)
stand_dev = 0
lst_of_results = np.array(lst_of_results)
for i in range(10):
    stand_dev = (lst_of_results[i] - mean_acc) ** 2
stand_dev = stand_dev / lst_of_results.shape[0]
```

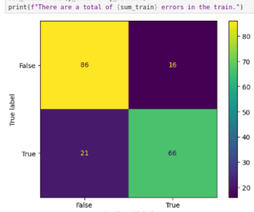
```
[139]: print(f'A acurácia média para 10 amostras é : {mean_acc * 100}% +- {stand_dev}')
```

```
A acurácia média para 10 amostras é : 64.07407407407408% +- 0.00012818167962200854
```

```
[140]: y_hatrain = train_test_ELM.test_ELM(X_train, Z, w, True)
```

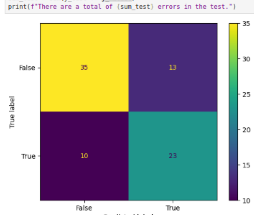
Utilizando com 30 neuronios obtemos os resultados :

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




```
There are a total of 37 errors in the train.
```

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



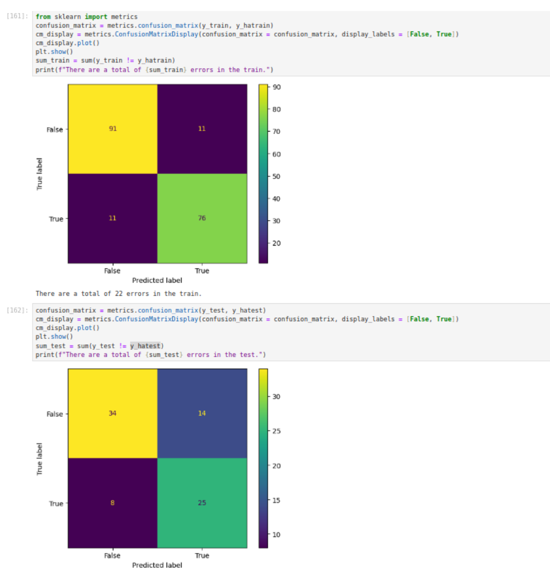

```
There are a total of 23 errors in the test.
```

```
[148]: import train_test_ELM
from sklearn.metrics import accuracy_score
p = 30
X_train = np.array(X_train)
X_test = np.array(X_test)
y_train = np.array(y_train)
y_test = np.array(y_test)
mean_acc = 0
lst_of_results = list()
for i in range(10):
    train_ELM = train_test_ELM.train_ELM(X_train, y_train, p, control = True)
    w = np.array(train_ELM[0])
    H = np.array(train_ELM[1])
    Z = np.array(train_ELM[2])
    y_hatest = train_test_ELM.test_ELM(X_test, Z, w, True)
    mean_acc += accuracy_score(y_test, y_hatest)
    lst_of_results.append(accuracy_score(y_test, y_hatest))
mean_acc = (mean_acc / 10)
stand_dev = 0
lst_of_results = np.array(lst_of_results)
for i in range(10):
    stand_dev = (lst_of_results[i] - mean_acc) ** 2
stand_dev = stand_dev / lst_of_results.shape[0]
```

```
[149]: print(f'A acurácia média para 10 amostras é : {mean_acc * 100}% +- {stand_dev}')
```

```
A acurácia média para 10 amostras é : 69.62962962962962% +- 3.901844231062364e-05
```

Utilizando com 50 neuronios obtemos os resultados :

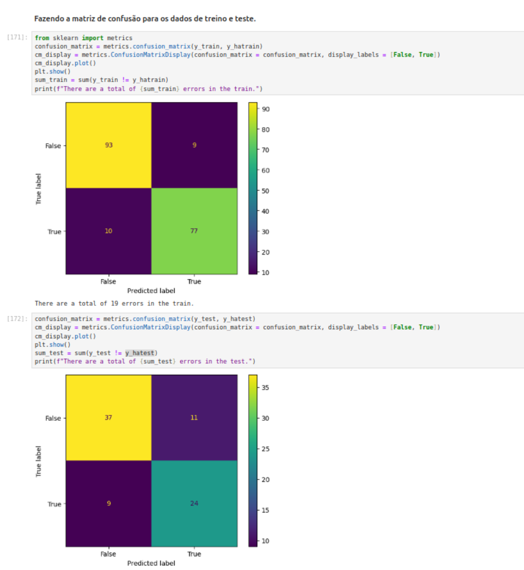


```
[157]: from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(df_statlog.iloc[ :, 0 : 13], df_statlog.iloc[ :, 13], random_state = 0, train_size = 0.7
```

```
[158]: import train_test_ELM
from sklearn.metrics import accuracy_score
p = 50
X_train = np.array(X_train)
X_test = np.array(X_test)
y_train = np.array(y_train)
y_test = np.array(y_test)
mean_acc = 0
lst_of_results = list()
for i in range(10):
    train_ELM = train_test_ELM.train_ELM(X_train, y_train, p, control = True)
    w = np.array(train_ELM[0])
    H = np.array(train_ELM[1])
    Z = np.array(train_ELM[2])
    y_hatest = train_test_ELM.test_ELM(X_test, Z, w, True)
    mean_acc += accuracy_score(y_test, y_hatest)
    lst_of_results.append(accuracy_score(y_test, y_hatest))
mean_acc = (mean_acc / 10)
stand_dev = 0
lst_of_results = np.array(lst_of_results)
for i in range(10):
    stand_dev = (lst_of_results[i] - mean_acc) ** 2
stand_dev = stand_dev / lst_of_results.shape[0]
```

```
[159]: print(f'A acurácia média para 10 amostras é : {mean_acc * 100}% +- {stand_dev}')

A acurácia média para 10 amostras é : 71.23456790123457% +- 2.575826855662248e-05
```

Utilizando com 100 neuronios obtemos os resultados :

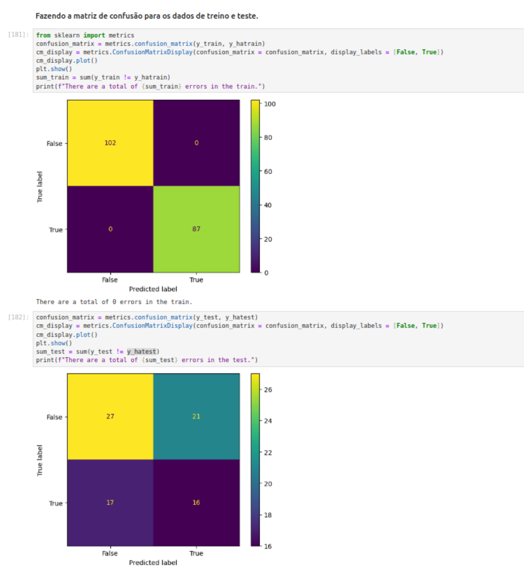


```
[168]: import train_test_ELM
from sklearn.metrics import accuracy_score
p = 100
X_train = np.array(X_train)
X_test = np.array(X_test)
y_train = np.array(y_train)
y_test = np.array(y_test)
mean_acc = 0
lst_of_results = list()
for i in range(10):
    train_ELM = train_test_ELM.train_ELM(X_train, y_train, p, control = True)
    w = np.array(train_ELM[0])
    H = np.array(train_ELM[1])
    Z = np.array(train_ELM[2])
    y_hatest = train_test_ELM.test_ELM(X_test, Z, w, True)
    mean_acc += accuracy_score(y_test, y_hatest)
    lst_of_results.append(accuracy_score(y_test, y_hatest))
mean_acc = (mean_acc / 10)
stand_dev = 0
lst_of_results = np.array(lst_of_results)
for i in range(10):
    stand_dev = (lst_of_results[i] - mean_acc) ** 2
stand_dev = stand_dev / lst_of_results.shape[0]
```

```
[169]: print(f'A acurácia média para 10 amostras é : {mean_acc * 100}% +- {stand_dev}')
```

```
A acurácia média para 10 amostras é : 75.92592592592592% +- 3.8103947568967114e-06
```

Utilizando com 300 neuronios obtemos os resultados :



[178]:
```python
import train_test_ELM
from sklearn.metrics import accuracy_score
p = 300
X_train = np.array(X_train)
X_test = np.array(X_test)
y_train = np.array(y_train)
y_test = np.array(y_test)
mean_acc = 0
lst_of_results = list()
for i in range(10):
    train_ELM = train_test_ELM.train_ELM(X_train, y_train, p, control = True)
    w = np.array(train_ELM[0])
    H = np.array(train_ELM[1])
    Z = np.array(train_ELM[2])
    y_hatest = train_test_ELM.test_ELM(X_test, Z, w, True)
    mean_acc += accuracy_score(y_test, y_hatest)
    lst_of_results.append(accuracy_score(y_test, y_hatest))
mean_acc = (mean_acc / 10)
stand_dev = 0
lst_of_results = np.array(lst_of_results)
for i in range(10):
    stand_dev = (lst_of_results[i] - mean_acc) ** 2
stand_dev = stand_dev / lst_of_results.shape[0]
```

[179]:
```python
print(f'A acurácia média para 10 amostras é : {mean_acc * 100}% +- {stand_dev}')
```

```
A acurácia média para 10 amostras é : 58.271604938271594% +- 0.0002688614540466377
```

Portanto, observando as matrizes de confusão e as acurácias médias, é possível afirmar que o modelo, ao usar 300 neuronios na camada intermediária é 1 modelo com overfitting. Portanto, o modelo com a melhor generalização é o modelo que contém 100 neurônios na camada intermediária.

Os dois exemplos de redes ELM mostram que os resultados da acurácia aumentam conforme o número de neuronios da camada intermediária. No entanto, há um certo limite de neurônios que fazem com que o modelo seja um overfitting. A ideia é ajustar corretamente o hyperparametro p para a boa generalização do modelo.

1.3 Perceptron

Agora, iremos utilizar o perceptron simples sobre os mesmos conjunto de dados, e avaliar a performance dele através da acurácia e a matriz de confusão. A função de treino do perceptron simples é mostrada abaixo :

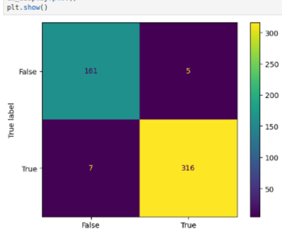
```
[82]: from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(Breast_Cancer_PANDAS_df.iloc[:, 1 : 10], Breast_Cancer_PANDAS_df.iloc[:, 10], random_st
```

```
[83]: def train_perceptron(x_inp : np.array, yi : np.array, learn_rate : float, tol : float, max_epochs : int, control_var : bool): # Fazendo o t
    try : # Caso eu tenha algum problema com as colunas do meu programa...
        n_rows = x_inp.shape[0]
        n_cols = x_inp.shape[1]
    except Exception as error:
        if error == "IndexError":
            print("Now, you don't have cols, so we will change it...\n")
            n_cols = 1
        else:
            print(f"The error {error} is hapenning \n")
            print("Breaking the program...")
            sys.exit()
    finally:
        if control_var == True: # control_var é 1 variável de controla que controlará quando usarei um certo threshold...
            w = (np.random.uniform(size = n_cols + 1) - 0.5) # Inicializando o pesos com o tamanho n_cols + 1.
            ones = np.ones((n_rows, 1))
            x_inp = np.concatenate((x_inp, ones), axis = 1) # Apenas colocando as colunas no vetor de entrada.
        else:
            w = (np.random.uniform(size = n_cols) - 0.5)
        n_epochs = 0
        err_epoch = tol + 1
        lst_errors = np.zeros((max_epochs))
        lst_outs = np.zeros((n_rows))
        aux = 0
        while ((n_epochs < max_epochs) and (err_epoch > tol)):
            error_grad = 0
            rand_order = np.random.permutation(n_rows)
            for i in range(n_rows):
                # Escolhendo uma entrada aleatória.
                i_rand = rand_order[i]
                x_val = x_inp[i_rand, :]
                y_hat = 1 if np.dot(x_val, w) >= 0 else 0 # A saída separadora do perceptron.
                err = (yi[i_rand] - y_hat)
                dw = (learn_rate*err*x_inp[i_rand, :])
                w = w + dw # Atualização de pesos.
                if n_epochs == max_epochs - 1:
                    lst_outs[aux] = y_hat
                    aux += 1
                error_grad = error_grad + (err**2)
            lst_errors[n_epochs] = error_grad / n_rows
            n_epochs += 1
    return (w, lst_errors, lst_outs)

def yperceptron(x_input : np.array, w : np.array, control_var : bool):
    try :
        n_rows = x_input.shape[0]
        n_cols = x_input.shape[1]
    except Exception as error:
        print(f"The error {error} is happening ...")
        n_cols = 1
        x_input = x_input.reshape(-1, 1)
    if control_var == True:
        ones = np.ones((n_rows, 1))
        x_input = np.concatenate((x_input, ones), axis = 1) # Apenas colocando as colunas no vetor de entrada.
    u = np.dot(x_input, w)

    y = np.where(u >= 0, 1, 0) # Compara elemento a elemento com 0, retorna 1 caso maior e 0 caso menor.
    return y
```

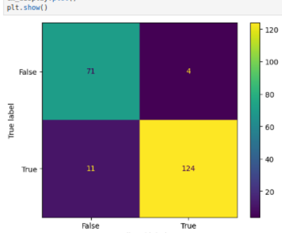
Os restultados do perceptron tanto para o Breast Cancer quanto para o Hearth Disease estão plotados abaixo :

Breast Cancer :

```
[88]: from sklearn import metrics
confusion_matrix = metrics.confusion_matrix(y_train, y_hat_train)
cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix = confusion_matrix, display_labels = [False, True])
cm_display.plot()
plt.show()
```




```
[89]: confusion_matrix = metrics.confusion_matrix(y_test, y_hat_test)
cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix = confusion_matrix, display_labels = [False, True])
cm_display.plot()
plt.show()
```



```
    y = np.where(u >= 0, 1, 0) # Compara elemento a elemento com 0, retorna 1 caso maior e 0 caso menor.
    return y
```

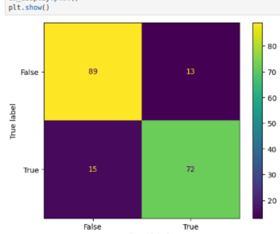
[84]:
```
from sklearn.metrics import accuracy_score
X_train = np.array(X_train)
X_test = np.array(X_test)
y_train = np.array(y_train)
y_test = np.array(y_test)
mean_acc = 0
lst_of_results = list()
for i in range(10):
    lst_return = train_perceptron(X_train, y_train, 0.01, 0.1, 10, True)
    w = lst_return[0]
    y_hat_test = yperceptron(X_test, w, True)
    lst_of_results.append(accuracy_score(y_test, y_hat_test))
    mean_acc += accuracy_score(y_test, y_hat_test)
mean_acc = (mean_acc / 10)
lst_of_results = np.array(lst_of_results)
stand_dev = 0
for i in range(10):
    stand_dev = (lst_of_results[i] - mean_acc) ** 2
stand_dev = stand_dev / lst_of_results.shape[0]
```

[90]:
```
print(f'A acurácia média para 10 amostras é : {mean_acc * 100}% +- {stand_dev}')
```

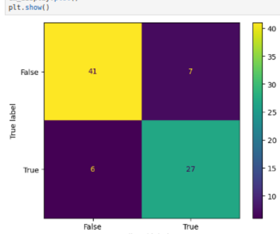
```
A acurácia média para 10 amostras é : 91.66666666666667% +- 1.417233560090693e-05
```

Hearth Disease :

[27]:
```
from sklearn import metrics
confusion_matrix = metrics.confusion_matrix(y_train, y_hat_train)
cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix = confusion_matrix, display_labels = [False, True])
cm_display.plot()
plt.show()
```



[28]:
```
confusion_matrix = metrics.confusion_matrix(y_test, y_hat_test)
cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix = confusion_matrix, display_labels = [False, True])
cm_display.plot()
plt.show()
```



[24]:
```
from sklearn.metrics import accuracy_score
X_train = np.array(X_train)
X_test = np.array(X_test)
y_train = np.array(y_train)
y_test = np.array(y_test)
mean_acc = 0
lst_of_results = list()
for i in range(10):
    lst_return = train_perceptron(X_train, y_train, 0.01, 0.1, 100, True)
    w = lst_return[0]
    y_hat_test = yperceptron(X_test, w, True)
    lst_of_results.append(accuracy_score(y_test, y_hat_test))
    mean_acc += accuracy_score(y_test, y_hat_test)
mean_acc = (mean_acc / 10)
lst_of_results = np.array(lst_of_results)
stand_dev = 0
for i in range(10):
    stand_dev = (lst_of_results[i] - mean_acc) ** 2
stand_dev = stand_dev / lst_of_results.shape[0]
```

[25]:
```
print(f'A acurácia média para 10 amostras é : {mean_acc * 100}% +- {stand_dev}')
```

```
A acurácia média para 10 amostras é : 78.64197530864197% +- 0.00028181679622008785
```

Após analisar tanto a acurácia do perceptron simples quanto a acurácia das redes ELM é possível concluir que o dataset Hearth Disease é um dataset com uma correlação mais complexa entre as variáveis, pois a acurácia é baixa mesmo variando diversas vezes os parâmetros da rede de modo a tentar melhorar a acurácia.