act_uresti

September 8, 2025

---

1 Actividad Modulo 2

Yose Miguel Sotomayor Carneado A0150908

1.1 Importamos librerias necesarias

[1]:
```
import numpy as np
import pandas as pd

from matplotlib import pyplot as plt
import seaborn as sns

from sklearn.metrics import ConfusionMatrixDisplay, classification_report
from sklearn.datasets import load_breast_cancer

from NNMultiClass import NNMultiClass, train_test_split_stratified,␣
↪transform_standardizer, fit_standardizer

%matplotlib inline
```

1.2 Data

[2]:
```
data = load_breast_cancer()

X = data.data
y = data.target
```

[3]:
```
df = pd.DataFrame(X, y, columns=data.feature_names)
df
```

[3]:
```
   mean radius  mean texture  mean perimeter  mean area  mean smoothness  \
0        17.99         10.38          122.80     1001.0          0.11840
0        20.57         17.77          132.90     1326.0          0.08474
```

0	19.69	21.25	130.00	1203.0	0.10960
0	11.42	20.38	77.58	386.1	0.14250
0	20.29	14.34	135.10	1297.0	0.10030
..	…	…	…	…	…
0	21.56	22.39	142.00	1479.0	0.11100
0	20.13	28.25	131.20	1261.0	0.09780
0	16.60	28.08	108.30	858.1	0.08455
0	20.60	29.33	140.10	1265.0	0.11780
1	7.76	24.54	47.92	181.0	0.05263

	mean compactness	mean concavity	mean concave points	mean symmetry	\
0	0.27760	0.30010	0.14710	0.2419	
0	0.07864	0.08690	0.07017	0.1812	
0	0.15990	0.19740	0.12790	0.2069	
0	0.28390	0.24140	0.10520	0.2597	
0	0.13280	0.19800	0.10430	0.1809	
..	…	…	…	…	
0	0.11590	0.24390	0.13890	0.1726	
0	0.10340	0.14400	0.09791	0.1752	
0	0.10230	0.09251	0.05302	0.1590	
0	0.27700	0.35140	0.15200	0.2397	
1	0.04362	0.00000	0.00000	0.1587	

	mean fractal dimension	…	worst radius	worst texture	worst perimeter	\
0	0.07871	…	25.380	17.33	184.60	
0	0.05667	…	24.990	23.41	158.80	
0	0.05999	…	23.570	25.53	152.50	
0	0.09744	…	14.910	26.50	98.87	
0	0.05883	…	22.540	16.67	152.20	
..	…	…	…	…	…	
0	0.05623	…	25.450	26.40	166.10	
0	0.05533	…	23.690	38.25	155.00	
0	0.05648	…	18.980	34.12	126.70	
0	0.07016	…	25.740	39.42	184.60	
1	0.05884	…	9.456	30.37	59.16	

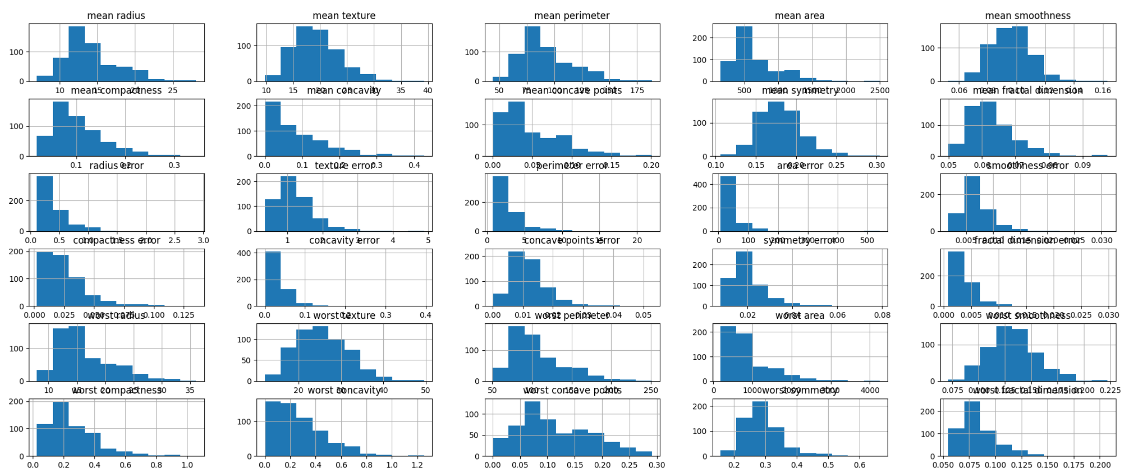
	worst area	worst smoothness	worst compactness	worst concavity	\
0	2019.0	0.16220	0.66560	0.7119	
0	1956.0	0.12380	0.18660	0.2416	
0	1709.0	0.14440	0.42450	0.4504	
0	567.7	0.20980	0.86630	0.6869	
0	1575.0	0.13740	0.20500	0.4000	
..	…	…	…	…	
0	2027.0	0.14100	0.21130	0.4107	
0	1731.0	0.11660	0.19220	0.3215	
0	1124.0	0.11390	0.30940	0.3403	
0	1821.0	0.16500	0.86810	0.9387	

```
1        268.6            0.08996            0.06444            0.0000

    worst concave points  worst symmetry  worst fractal dimension
0                 0.2654          0.4601                  0.11890
0                 0.1860          0.2750                  0.08902
0                 0.2430          0.3613                  0.08758
0                 0.2575          0.6638                  0.17300
0                 0.1625          0.2364                  0.07678
..                   …               …                       …
0                 0.2216          0.2060                  0.07115
0                 0.1628          0.2572                  0.06637
0                 0.1418          0.2218                  0.07820
0                 0.2650          0.4087                  0.12400
1                 0.0000          0.2871                  0.07039

[569 rows x 30 columns]
```

[4]:
```python
df.hist(figsize=(25, 10))
plt.show()
```



vemos que las distribuciones son bastante validas, y no hay inconsistencia en los datos.

1.2.1 Train-Test Split

[5]:
```python
X_train, X_test, y_train, y_test = train_test_split_stratified(X, y,
    test_size=0.2, seed=42)
```

1.2.2 Scale the data

se escalan los datos para que la red neuronal no tenga problemas con numeros grandes

[6]:
```
X_test_scaled  = transform_standardizer(X_test, *fit_standardizer(X_train))
X_train_scaled = transform_standardizer(X_train, *fit_standardizer(X_train))
```

1.3 Neural Network Configuration

[7]:
```
input_size = X_test_scaled.shape[1]
output_size = len(np.unique(y))
layers = 64

layer_sizes = [input_size] + [layers] + [output_size]

nn = NNMultiClass(layer_sizes=layer_sizes, hidden_activation="relu", seed=42,
 ↪lr=3e-1)
nn.show_weights()
y_pred = nn.predict(X_test_scaled)
```

```
Pesos capa 0 (30 → 64):
[[ 0.07867761 -0.26852274  0.19376567 … -0.08646694  0.04202266
   0.15136196]
 [ 0.18363791  0.20484138 -0.09004043 …  0.09214371  0.37782318
  -0.3069373 ]
 [-0.16518314 -0.23924088 -0.10064846 …  0.42626776  0.44504858
  -0.04635166]
 …
 [ 0.0189196   0.00968846 -0.02346277 …  0.2622049  -0.4281804
  -0.00204094]
 [ 0.0707058  -0.07765708 -0.47822613 …  0.17353917  0.12267754
  -0.17865651]
 [-0.0435457   0.26949339  0.10156947 …  0.26474372 -0.08192308
  -0.37856759]]

Pesos capa 1 (64 → 2):
[[-0.13671713  0.12836322]
 [-0.24018807  0.14112389]
 [-0.00688776 -0.14904942]
 [-0.07288938 -0.12715304]
 [ 0.10827815  0.09155348]
 [-0.21489514 -0.05044901]
 [-0.10789821  0.16168875]
 [-0.16876239  0.20052864]
 [-0.00416184 -0.1452418 ]
 [-0.08465927 -0.10020236]
```

```
[-0.1593273   0.01185336]
[-0.15348944 -0.27254578]
[-0.04745956 -0.22535702]
[-0.09402011 -0.11363786]
[-0.10890201  0.25012039]
[ 0.00705145  0.03067138]
[-0.22204979 -0.08174539]
[ 0.02691283 -0.12806611]
[ 0.11350786 -0.15033707]
[-0.03579577 -0.07325645]
[ 0.02054008 -0.08006126]
[-0.0338617  -0.02951626]
[-0.09179847  0.06063673]
[ 0.02979855 -0.01107674]
[-0.05380729  0.099568  ]
[-0.02947928 -0.03833716]
[ 0.01308625  0.21207334]
[-0.16754453  0.09623211]
[-0.2366464   0.02715914]
[ 0.19688535  0.06108357]
[ 0.02132813 -0.07005618]
[ 0.0038809  -0.03150785]
[-0.09824708 -0.03623445]
[ 0.02479765 -0.09590213]
[-0.10078523 -0.17003043]
[ 0.04884063  0.06816541]
[ 0.02895756  0.12064631]
[ 0.01098345  0.04900137]
[ 0.03797968 -0.0114425 ]
[-0.06635273  0.27676908]
[-0.05649387 -0.08323471]
[ 0.05425123  0.0314818 ]
[-0.17559894  0.14033478]
[-0.01177432 -0.13886639]
[ 0.14859801  0.07820945]
[-0.15806775  0.22214786]
[ 0.08234967 -0.00127451]
[ 0.19877735  0.01991222]
[-0.00351038 -0.08253751]
[ 0.0727313   0.05957537]
[ 0.00183502 -0.09874829]
[ 0.01624123 -0.00872485]
[-0.16668429 -0.18397654]
[ 0.07064417 -0.32523604]
[ 0.02567632 -0.05976038]
[ 0.16353083 -0.06168353]
[ 0.08243553 -0.04961423]
[ 0.01421852  0.09210457]
```

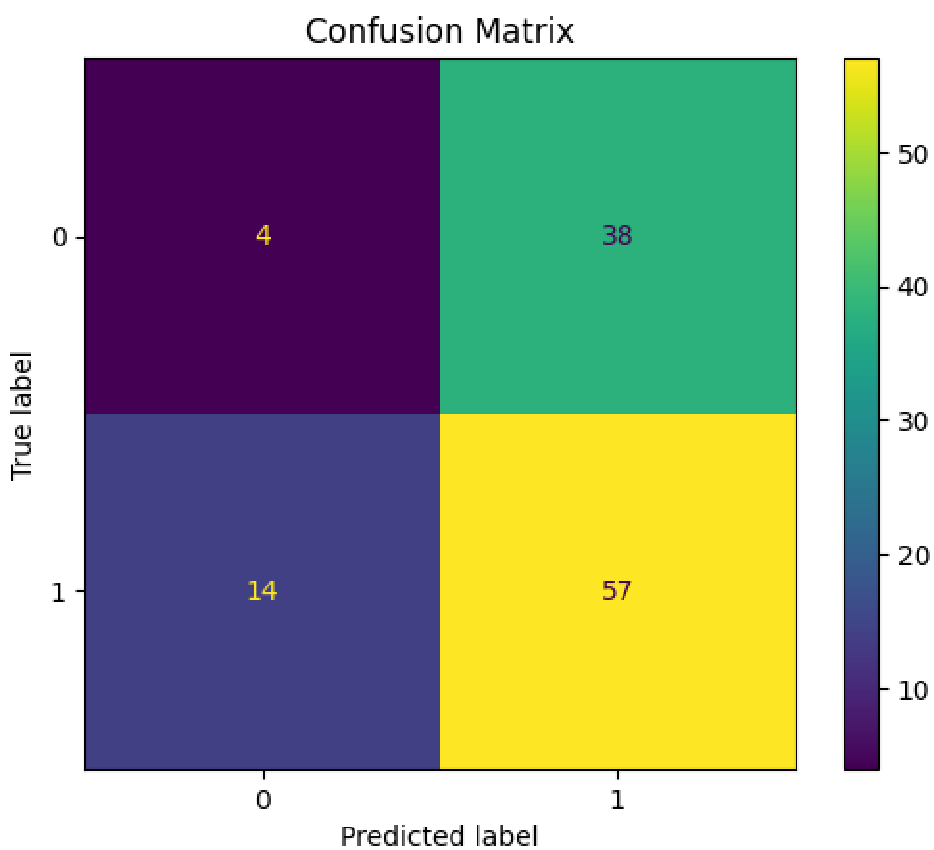
```
 [-0.05781946  0.12456936]
 [-0.12421223  0.03682411]
 [ 0.11532997  0.08250469]
 [ 0.23955561 -0.09542991]
 [-0.17239366 -0.12429189]
 [ 0.08255532  0.06224037]]
```

1.4 Pre - Backpropagation Prediction

1.4.1 Confusion Matrix

[8]:

```python
ConfusionMatrixDisplay.from_predictions(y_test, y_pred)
plt.title("Confusion Matrix")
plt.show()
```



1.4.2 Classification Report

[9]:
```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.22      0.10      0.13        42
           1       0.60      0.80      0.69        71

    accuracy                           0.54       113
   macro avg       0.41      0.45      0.41       113
weighted avg       0.46      0.54      0.48       113
```

Al tener la red neuronal con pesos aleatorios, nos damos cuenta que el modelo es muy malo, no acertando en nada excepto en el caso del recall para cuando si hay cancer.

1.5 Post - Backpropagation Prediction

[10]:
```
nn.fit(X_train_scaled, y_train, epochs=100, verbose=True, batch_size=12)
nn.show_weights()
y_pred_back = nn.predict(X_test_scaled)
```

```
Epoch    1 | loss=0.0754 | acc=0.9627
Epoch   10 | loss=0.0118 | acc=0.9978
Epoch   20 | loss=0.0044 | acc=1.0000
Epoch   30 | loss=0.0025 | acc=1.0000
Epoch   40 | loss=0.0018 | acc=1.0000
Epoch   50 | loss=0.0012 | acc=1.0000
Epoch   60 | loss=0.0010 | acc=1.0000
Epoch   70 | loss=0.0008 | acc=1.0000
Epoch   80 | loss=0.0007 | acc=1.0000
Epoch   90 | loss=0.0006 | acc=1.0000
Epoch  100 | loss=0.0005 | acc=1.0000

Pesos capa 0 (30 → 64):
[[ 0.09582232 -0.24182964  0.1429559  … -0.00692755 -0.0270322
   0.09307529]
 [ 0.06365033  0.31622577 -0.08647731 …  0.03312427  0.26522226
  -0.33071288]
 [-0.16502683 -0.20602504 -0.1481132  …  0.50929718  0.377776
  -0.10055337]
 …
 [-0.10474993 -0.13614516  0.02158844 …  0.35400317 -0.34604035
   0.04677166]
 [ 0.01184807 -0.22272733 -0.46169803 …  0.12743401  0.16537303
  -0.12167377]
```

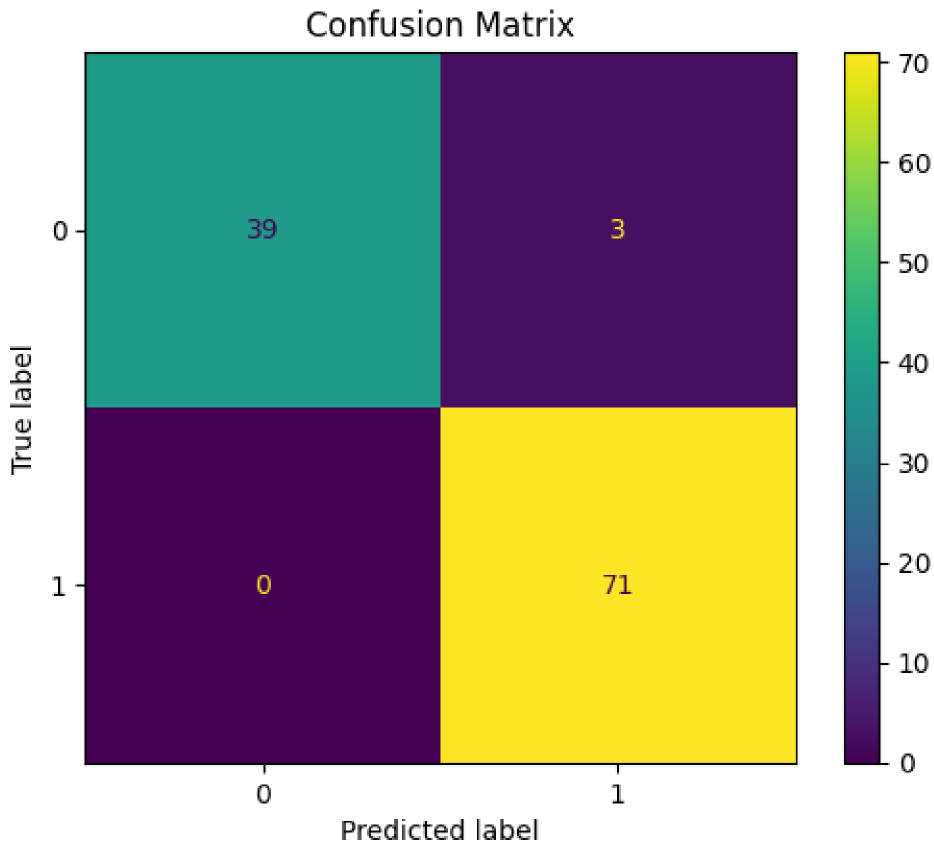
```
 [-0.37771899  0.34937802  0.18190729 …  0.37980639  0.04488426
  -0.30202109]]

Pesos capa 1 (64 → 2):
[[-0.4238707   0.41551678]
 [-0.46451026  0.36544608]
 [ 0.26965039 -0.42558757]
 [-0.06563598 -0.13440645]
 [ 0.75524641 -0.55541478]
 [-0.11586743 -0.14947673]
 [-0.26339052  0.31718106]
 [-0.30631175  0.338078  ]
 [ 0.68701407 -0.83641771]
 [ 0.17469264 -0.35955427]
 [ 0.38974796 -0.5372219 ]
 [-0.05085441 -0.37518081]
 [-0.22963684 -0.04317975]
 [-0.5455276   0.33786963]
 [ 0.31542774 -0.17420936]
 [ 0.00232824  0.03539458]
 [-0.66155474  0.35775956]
 [ 0.05576089 -0.15691418]
 [ 0.09044808 -0.1272773 ]
 [ 0.38257371 -0.49162593]
 [ 0.11920511 -0.17872628]
 [-0.11949055  0.05611259]
 [-0.11667327  0.08551153]
 [-0.13493247  0.15365427]
 [-0.3141884   0.35994911]
 [-0.20827038  0.14045394]
 [-0.93786824  1.16302783]
 [-0.52375069  0.45243828]
 [-0.54494272  0.33545547]
 [-0.30694987  0.56491879]
 [-0.34925877  0.30053072]
 [ 0.30763542 -0.33526237]
 [-0.57051047  0.43602894]
 [ 0.47927616 -0.55038065]
 [-0.32807945  0.0572638 ]
 [ 0.9033696  -0.78636357]
 [ 0.57043922 -0.42083536]
 [-0.10346656  0.16345138]
 [ 0.24143225 -0.21489507]
 [-0.16817483  0.37859118]
 [-0.72486934  0.58514076]
 [ 0.77407554 -0.68834251]
 [-0.52308219  0.48781802]
 [-0.41996018  0.26931947]
```

```
 [-0.37516139  0.60196885]
 [-0.48078672  0.54486682]
 [ 0.07780546  0.00326969]
 [ 0.37527923 -0.15658965]
 [-0.17469371  0.08864581]
 [-0.2435533   0.37585998]
 [-0.24643117  0.1495179 ]
 [-0.03237395  0.03989033]
 [ 0.47911329 -0.82977412]
 [ 0.44209931 -0.69669117]
 [-0.02922146 -0.00486261]
 [-0.23066274  0.33251004]
 [-0.02470126  0.05752256]
 [ 0.1993498  -0.09302672]
 [ 0.60658938 -0.53983948]
 [-0.02558089 -0.06180724]
 [ 0.24890958 -0.05107493]
 [ 0.25264623 -0.10852054]
 [ 0.12279248 -0.41947803]
 [ 0.36010641 -0.21531072]]
```

1.5.1 Confusion Matrix

[11]:
```
ConfusionMatrixDisplay.from_predictions(y_test, y_pred_back)
plt.title("Confusion Matrix")
plt.show()
```



1.5.2 Classification Report

[12]:
```
print(classification_report(y_test, y_pred_back))
```

```
              precision    recall  f1-score   support

           0       1.00      0.93      0.96        42
           1       0.96      1.00      0.98        71

    accuracy                           0.97       113
   macro avg       0.98      0.96      0.97       113
weighted avg       0.97      0.97      0.97       113
```

Al aplicar el algoritmo de backpropagation observamos que el modelo funciona muy bien, dandonos resultados consistentes y válidos, dandonos a entender que el modelo aprendio bien los datos y se comporta bien con los datos de prueba.

---

2 Conclusiones

En resumen, el trabajo realizado permitió consolidar aprendizajes clave, demostrar avances técnicos y prácticos, y fortalecer la capacidad de análisis y adaptación.

Aunque se identificaron áreas de mejora que servirán de guía para el perfeccionamiento futuro, se cuenta ya con una base sólida que permitirá afrontar con mayor claridad y seguridad los retos de las siguientes etapas.