Jwean Dabre

RollNo 13

SE-AIML

Date: 11/02/2026

Experiment 5: Disease Prediction using Naive Bayes and Neural Network with Comparison of Classifiers

In [3]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, precision_score,recal
```

Load the dataset

In [4]:
```
data=load_breast_cancer()
X=data.data
y=data.target
print("Classes:",data.target_names)
```

```
Classes: ['malignant' 'benign']
```

In [5]:
```
df = pd.DataFrame(data.data, columns=data.feature_names)
df["target"] = data.target
```

In [6]:
```
print(df.shape)
```

```
(569, 31)
```

In [7]:
```
print(df.head())
```

```
   mean radius  mean texture  mean perimeter  mean area  mean smoothness  \
0        17.99         10.38          122.80     1001.0          0.11840
1        20.57         17.77          132.90     1326.0          0.08474
2        19.69         21.25          130.00     1203.0          0.10960
3        11.42         20.38           77.58      386.1          0.14250
4        20.29         14.34          135.10     1297.0          0.10030

   mean compactness  mean concavity  mean concave points  mean symmetry  \
0           0.27760          0.3001              0.14710         0.2419
1           0.07864          0.0869              0.07017         0.1812
2           0.15990          0.1974              0.12790         0.2069
3           0.28390          0.2414              0.10520         0.2597
4           0.13280          0.1980              0.10430         0.1809

   mean fractal dimension  ...  worst texture  worst perimeter  worst area  \
0                 0.07871  ...          17.33           184.60      2019.0
1                 0.05667  ...          23.41           158.80      1956.0
2                 0.05999  ...          25.53           152.50      1709.0
3                 0.09744  ...          26.50            98.87       567.7
4                 0.05883  ...          16.67           152.20      1575.0

   worst smoothness  worst compactness  worst concavity  worst concave points  \
0            0.1622             0.6656           0.7119                0.2654
1            0.1238             0.1866           0.2416                0.1860
2            0.1444             0.4245           0.4504                0.2430
3            0.2098             0.8663           0.6869                0.2575
4            0.1374             0.2050           0.4000                0.1625

   worst symmetry  worst fractal dimension  target
0          0.4601                  0.11890       0
1          0.2750                  0.08902       0
2          0.3613                  0.08758       0
3          0.6638                  0.17300       0
4          0.2364                  0.07678       0

[5 rows x 31 columns]
```

In [8]:
```
print(df.tail())
```

```
     mean radius  mean texture  mean perimeter  mean area  mean smoothness  \
564        21.56         22.39          142.00     1479.0          0.11100
565        20.13         28.25          131.20     1261.0          0.09780
566        16.60         28.08          108.30      858.1          0.08455
567        20.60         29.33          140.10     1265.0          0.11780
568         7.76         24.54           47.92      181.0          0.05263

     mean compactness  mean concavity  mean concave points  mean symmetry  \
564           0.11590         0.24390              0.13890         0.1726
565           0.10340         0.14400              0.09791         0.1752
566           0.10230         0.09251              0.05302         0.1590
567           0.27700         0.35140              0.15200         0.2397
568           0.04362         0.00000              0.00000         0.1587

     mean fractal dimension  ...  worst texture  worst perimeter  worst area  \
564                 0.05623  ...          26.40           166.10      2027.0
565                 0.05533  ...          38.25           155.00      1731.0
566                 0.05648  ...          34.12           126.70      1124.0
567                 0.07016  ...          39.42           184.60      1821.0
568                 0.05884  ...          30.37            59.16       268.6

     worst smoothness  worst compactness  worst concavity  \
564           0.14100            0.21130           0.4107
565           0.11660            0.19220           0.3215
566           0.11390            0.30940           0.3403
567           0.16500            0.86810           0.9387
568           0.08996            0.06444           0.0000

     worst concave points  worst symmetry  worst fractal dimension  target
564                0.2216          0.2060                  0.07115       0
565                0.1628          0.2572                  0.06637       0
566                0.1418          0.2218                  0.07820       0
567                0.2650          0.4087                  0.12400       0
568                0.0000          0.2871                  0.07039       1

[5 rows x 31 columns]
```

In [9]:
```
print("\nDataset Info:")
print(df.info())
```

```
Dataset Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 31 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   mean radius              569 non-null    float64
 1   mean texture             569 non-null    float64
 2   mean perimeter           569 non-null    float64
 3   mean area                569 non-null    float64
 4   mean smoothness          569 non-null    float64
 5   mean compactness         569 non-null    float64
 6   mean concavity           569 non-null    float64
 7   mean concave points      569 non-null    float64
 8   mean symmetry            569 non-null    float64
 9   mean fractal dimension   569 non-null    float64
 10  radius error             569 non-null    float64
 11  texture error            569 non-null    float64
 12  perimeter error          569 non-null    float64
 13  area error               569 non-null    float64
 14  smoothness error         569 non-null    float64
 15  compactness error        569 non-null    float64
 16  concavity error          569 non-null    float64
 17  concave points error     569 non-null    float64
 18  symmetry error           569 non-null    float64
 19  fractal dimension error  569 non-null    float64
 20  worst radius             569 non-null    float64
 21  worst texture            569 non-null    float64
 22  worst perimeter          569 non-null    float64
 23  worst area               569 non-null    float64
 24  worst smoothness         569 non-null    float64
 25  worst compactness        569 non-null    float64
 26  worst concavity          569 non-null    float64
 27  worst concave points     569 non-null    float64
 28  worst symmetry           569 non-null    float64
 29  worst fractal dimension  569 non-null    float64
 30  target                   569 non-null    int64  
dtypes: float64(30), int64(1)
memory usage: 137.9 KB
None
```

In [10]:
```
print("\ndescribe:")
print(df.describe())
```

describe:

	mean radius	mean texture	mean perimeter	mean area
count	569.000000	569.000000	569.000000	569.000000
mean	14.127292	19.289649	91.969033	654.889104
std	3.524049	4.301036	24.298981	351.914129
min	6.981000	9.710000	43.790000	143.500000
25%	11.700000	16.170000	75.170000	420.300000
50%	13.370000	18.840000	86.240000	551.100000
75%	15.780000	21.800000	104.100000	782.700000
max	28.110000	39.280000	188.500000	2501.000000

	mean smoothness	mean compactness	mean concavity	mean concave points
count	569.000000	569.000000	569.000000	569.000000
mean	0.096360	0.104341	0.088799	0.048919
std	0.014064	0.052813	0.079720	0.038803
min	0.052630	0.019380	0.000000	0.000000
25%	0.086370	0.064920	0.029560	0.020310
50%	0.095870	0.092630	0.061540	0.033500
75%	0.105300	0.130400	0.130700	0.074000
max	0.163400	0.345400	0.426800	0.201200

	mean symmetry	mean fractal dimension	...	worst texture
count	569.000000	569.000000	...	569.000000
mean	0.181162	0.062798	...	25.677223
std	0.027414	0.007060	...	6.146258
min	0.106000	0.049960	...	12.020000
25%	0.161900	0.057700	...	21.080000
50%	0.179200	0.061540	...	25.410000
75%	0.195700	0.066120	...	29.720000
max	0.304000	0.097440	...	49.540000

	worst perimeter	worst area	worst smoothness	worst compactness
count	569.000000	569.000000	569.000000	569.000000
mean	107.261213	880.583128	0.132369	0.254265
std	33.602542	569.356993	0.022832	0.157336
min	50.410000	185.200000	0.071170	0.027290
25%	84.110000	515.300000	0.116600	0.147200
50%	97.660000	686.500000	0.131300	0.211900
75%	125.400000	1084.000000	0.146000	0.339100
max	251.200000	4254.000000	0.222600	1.058000

	worst concavity	worst concave points	worst symmetry
count	569.000000	569.000000	569.000000
mean	0.272188	0.114606	0.290076
std	0.208624	0.065732	0.061867
min	0.000000	0.000000	0.156500
25%	0.114500	0.064930	0.250400
50%	0.226700	0.099930	0.282200
75%	0.382900	0.161400	0.317900
max	1.252000	0.291000	0.663800

	worst fractal dimension	target
count	569.000000	569.000000
mean	0.083946	0.627417
std	0.018061	0.483918
min	0.055040	0.000000

```
25%                   0.071460    0.000000
50%                   0.080040    1.000000
75%                   0.092080    1.000000
max                   0.207500    1.000000

[8 rows x 31 columns]
```

Split data

In [11]:
```
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.3)
```

Table A :-

Logistic Regression

In [12]:
```
lr=LogisticRegression(max_iter=5000)
lr.fit(X_train,y_train)
pred_lr=lr.predict(X_test)
print("Logistic Test Accuracy:",accuracy_score(y_test,pred_lr))
pred_lr_train=lr.predict(X_train)
print("Logistic Train Accuracy:",accuracy_score(y_train,pred_lr_train))
```

```
Logistic Test Accuracy: 0.9239766081871345
Logistic Train Accuracy: 0.9723618090452262
```

Decision Tree

In [13]:
```
dt=DecisionTreeClassifier()
dt.fit(X_train,y_train)
pred_dt=dt.predict(X_test)
print("Logistic Test Accuracy:",accuracy_score(y_test,pred_dt))
pred_dt_train=dt.predict(X_train)
print("Logistic Train Accuracy:",accuracy_score(y_train,pred_dt_train))
```

```
Logistic Test Accuracy: 0.9181286549707602
Logistic Train Accuracy: 1.0
```

KNN

In [14]:
```
knn=KNeighborsClassifier()
knn.fit(X_train,y_train)
pred_knn=knn.predict(X_test)
print("Logistic Test Accuracy:",accuracy_score(y_test,pred_knn))
pred_knn_train=knn.predict(X_train)
print("Logistic Train Accuracy:",accuracy_score(y_train,pred_knn_train))
```

```
Logistic Test Accuracy: 0.9122807017543859
Logistic Train Accuracy: 0.9597989949748744
```

Naive Bayes

In [15]:
```
nb=GaussianNB()
nb.fit(X_train,y_train)
pred_nb=nb.predict(X_test)
print("Logistic Test Accuracy:",accuracy_score(y_test,pred_nb))
pred_nb_train=nb.predict(X_train)
print("Logistic Train Accuracy:",accuracy_score(y_train,pred_nb_train))
```

```
Logistic Test Accuracy: 0.9181286549707602
Logistic Train Accuracy: 0.949748743718593
```

Neural Network

In [17]:
```
nn=MLPClassifier(max_iter=5000)
nn.fit(X_train,y_train)
pred_nn=nn.predict(X_test)
print("Logistic Test Accuracy:",accuracy_score(y_test,pred_nn))
pred_nn_train=nn.predict(X_train)
print("Logistic Train Accuracy:",accuracy_score(y_train,pred_nn_train))
```

```
Logistic Test Accuracy: 0.9298245614035088
Logistic Train Accuracy: 0.964824120603015
```

Table B

Logistic Regression

In [18]:
```
cm_lr=confusion_matrix(y_test,pred_lr)
print("\nLogistic Regression Confusion Matrix:")
print(cm_lr)

TN1, FP1, FN1, TP1 = cm_lr.ravel()
type1_lr=FP1/(FP1+TN1)
type2_lr=FN1/(FN1+TP1)

print("FP Type1:",FP1)
print("FN Type2:",FN1)
print("Type1 Error rate:",type1_lr*100)
print("Type2 Error rate: ",type2_lr*100)
```

```
Logistic Regression Confusion Matrix:
[[63  7]
 [ 6 95]]
FP Type1: 7
FN Type2: 6
Type1 Error rate: 10.0
Type2 Error rate:  5.9405940594059405
```

Decision Tree

In [19]:
```
cm_dt=confusion_matrix(y_test,pred_dt)
print("\nDecision tree Confusion Matrix:")
print(cm_dt)

TN2, FP2, FN2, TP2 = cm_dt.ravel()
type1_dt=FP2/(FP2+TN2)
type2_dt=FN2/(FN2+TP2)

print("FP Type1:",FP2)
print("FN Type2:",FN2)
print("Type1 Error rate:",type1_dt*100)
print("Type2 Error rate: ",type2_dt*100)
```

```
Decision tree Confusion Matrix:
[[62  8]
 [ 6 95]]
FP Type1: 8
FN Type2: 6
Type1 Error rate: 11.428571428571429
Type2 Error rate:  5.9405940594059405
```

KNN

In [20]:
```
cm_knn=confusion_matrix(y_test,pred_knn)
print("\nKNN Confusion Matrix:")
print(cm_knn)

TN3, FP3, FN3, TP3 = cm_knn.ravel()
type1_knn=FP3/(FP3+TN3)
type2_knn=FN3/(FN3+TP3)

print("FP Type1:",FP3)
print("FN Type2:",FN3)
print("Type1 Error rate:",type1_knn*100)
print("Type2 Error rate: ",type2_knn*100)
```

```
KNN Confusion Matrix:
[[ 56  14]
 [  1 100]]
FP Type1: 14
FN Type2: 1
Type1 Error rate: 20.0
Type2 Error rate:  0.9900990099009901
```

Naive Bayes

In [21]:
```
cm_nb=confusion_matrix(y_test,pred_nb)
print("\nNaive Bayes Confusion Matrix:")
print(cm_nb)
```

```
TN4, FP4, FN4, TP4 = cm_nb.ravel()
type1_nb=FP4/(FP4+TN4)
type2_nb=FN4/(FN4+TP4)

print("FP Type1:",FP4)
print("FN Type2:",FN4)
print("Type1 Error rate:",type1_nb*100)
print("Type2 Error rate: ",type2_nb*100)
```

```
Naive Bayes Confusion Matrix:
[[62  8]
 [ 6 95]]
FP Type1: 8
FN Type2: 6
Type1 Error rate: 11.428571428571429
Type2 Error rate:  5.9405940594059405
```

Neural Network

In [22]:
```
cm_nn=confusion_matrix(y_test,pred_nn)
print("\nNeural Network Confusion Matrix:")
print(cm_nn)

TN5, FP5, FN5, TP5 = cm_nn.ravel()
type1_nn=FP5/(FP5+TN5)
type2_nn=FN5/(FN5+TP5)

print("FP Type1:",FP5)
print("FN Type2:",FN5)
print("Type1 Error rate:",type1_nn*100)
print("Type2 Error rate: ",type2_nn*100)
```

```
Neural Network Confusion Matrix:
[[60 10]
 [ 2 99]]
FP Type1: 10
FN Type2: 2
Type1 Error rate: 14.285714285714285
Type2 Error rate:  1.9801980198019802
```

Table C

Logistic Regression

In [23]:
```
precision2= precision_score(y_test, pred_lr)
recall2= recall_score(y_test, pred_lr)
f12= f1_score(y_test, pred_lr)
roc2 = roc_auc_score(y_test, pred_lr)

print("Precision :", precision2)
print("Recall:", recall2)
print("Fi-score :", f12)
print("ROC-AUC:", roc2)
```

```
Precision : 0.9313725490196079
Recall: 0.9405940594059405
Fi-score : 0.9359605911330049
ROC-AUC: 0.9202970297029703
```

Decision Tree

In [24]:
```
precision1= precision_score(y_test, pred_dt)
recall1= recall_score(y_test, pred_dt)
f11= f1_score(y_test, pred_dt)
roc1 = roc_auc_score(y_test, pred_dt)

print("Precision :", precision1)
print("Recall:", recall1)
print("Fi-score :", f11)
print("ROC-AUC:", roc1)
```

```
Precision : 0.9223300970873787
Recall: 0.9405940594059405
Fi-score : 0.9313725490196079
ROC-AUC: 0.9131541725601131
```

KNN

In [25]:
```
precision3= precision_score(y_test, pred_knn)
recall3= recall_score(y_test, pred_knn)
f13= f1_score(y_test, pred_knn)
roc3 = roc_auc_score(y_test, pred_knn)

print("Precision :", precision3)
print("Recall:", recall3)
print("Fi-score :", f13)
print("ROC-AUC:", roc3)
```

```
Precision : 0.8771929824561403
Recall: 0.9900990099009901
Fi-score : 0.9302325581395349
ROC-AUC: 0.8950495049504951
```

Naive Bayes

In [26]:
```
precision4= precision_score(y_test, pred_nb)
recall4= recall_score(y_test, pred_nn)
f14= f1_score(y_test, pred_nn)
roc4 = roc_auc_score(y_test, pred_nn)

print("Precision :", precision4)
print("Recall:", recall4)
print("Fi-score :", f14)
print("ROC-AUC:", roc4)
```

```
Precision : 0.9223300970873787
Recall: 0.9801980198019802
Fi-score : 0.9428571428571428
ROC-AUC: 0.9186704384724188
```

Neural Network

In [27]:
```
precision5= precision_score(y_test, pred_nn)
recall5= recall_score(y_test, pred_nn)
f15= f1_score(y_test, pred_nn)
roc5 = roc_auc_score(y_test, pred_nn)

print("Precision :", precision5)
print("Recall:", recall5)
print("Fi-score :", f15)
print("ROC-AUC:", roc5)
```

```
Precision : 0.908256880733945
Recall: 0.9801980198019802
Fi-score : 0.9428571428571428
ROC-AUC: 0.9186704384724188
```

In [ ]:
```
```