ML-Exp 5: Disease Prediction using Naive Bayes and Neural Network with Comparison of Classifiers.

Student Name: Ben Dabre

Roll No.: 12 Batch: 1 Date: 11-02-2026

```
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 mean_squared_error,r2_score,precision_score,accuracy_score,recall_score,f1_score,roc_auc_score,confus
import pandas as pd
```

```
data=load_breast_cancer()

x=data.data
y=data.target
print("Classes:", data.target_names)

df=pd.DataFrame(x, columns=data.feature_names)
df['target']=y

df.head()
df.tail()
df.info()
df.columns
df.describe()
```

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

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	..
count	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	
mean	14.127292	19.289649	91.969033	654.889104	0.096360	0.104341	0.088799	0.048919	0.181162	0.062798	
std	3.524049	4.301036	24.298981	351.914129	0.014064	0.052813	0.079720	0.038803	0.027414	0.007060	
min	6.981000	9.710000	43.790000	143.500000	0.052630	0.019380	0.000000	0.000000	0.106000	0.049960	
25%	11.700000	16.170000	75.170000	420.300000	0.086370	0.064920	0.029560	0.020310	0.161900	0.057700	
50%	13.370000	18.840000	86.240000	551.100000	0.095870	0.092630	0.061540	0.033500	0.179200	0.061540	
75%	15.780000	21.800000	104.100000	782.700000	0.105300	0.130400	0.130700	0.074000	0.195700	0.066120	
max	28.110000	39.280000	188.500000	2501.000000	0.163400	0.345400	0.426800	0.201200	0.304000	0.097440	

8 rows × 31 columns

```
x_train, x_test, y_train, y_test=train_test_split(x,y,test_size=0.3)
```

```
lr = LogisticRegression(max_iter=5000)
lr.fit(x_train, y_train)
pred_lr = lr.predict(x_test)
print("Logistic TestAccuracy:", accuracy_score(y_test, pred_lr))
pred_lr_train = lr.predict(x_train)
print("Logistic TrainAccuracy:", accuracy_score(y_train, pred_lr_train))
```

```
Logistic TestAccuracy: 0.9590643274853801
Logistic TrainAccuracy: 0.9547738693467337
```

```
dt = DecisionTreeClassifier()
dt.fit(x_train, y_train)
pred_dt = dt.predict(x_test)
print("DecisionTreeClassifier TestAccuracy:", accuracy_score(y_test, pred_dt))
pred_dt_train = dt.predict(x_train)
print("DecisionTreeClassifier TrainAccuracy:", accuracy_score(y_train, pred_dt_train))
```

```
DecisionTreeClassifier TestAccuracy: 0.9415204678362573
DecisionTreeClassifier TrainAccuracy: 1.0
```

```
kn = KNeighborsClassifier()
kn.fit(x_train, y_train)
pred_kn = kn.predict(x_test)
print("KNeighborsClassifier TestAccuracy:", accuracy_score(y_test, pred_kn))
pred_kn_train = kn.predict(x_train)
print("KNeighborsClassifier TrainAccuracy:", accuracy_score(y_train, pred_kn_train))
```

```
KNeighborsClassifier TestAccuracy: 0.9181286549707602
KNeighborsClassifier TrainAccuracy: 0.9522613065326633
```

```
nb = GaussianNB()
nb.fit(x_train, y_train)
pred_nb = nb.predict(x_test)
print("GaussianNB TestAccuracy:", accuracy_score(y_test, pred_nb))
pred_nb_train = nb.predict(x_train)
print("GaussianNB TrainAccuracy:", accuracy_score(y_train, pred_nb_train))
```

```
GaussianNB TestAccuracy: 0.9122807017543859
GaussianNB TrainAccuracy: 0.949748743718593
```

```
mlp = MLPClassifier()
mlp.fit(x_train, y_train)
pred_mlp = mlp.predict(x_test)
print("MLPClassifier TestAccuracy:", accuracy_score(y_test, pred_mlp))
pred_mlp_train = mlp.predict(x_train)
print("MLPClassifier TrainAccuracy:", accuracy_score(y_train, pred_mlp_train))
```

```
MLPClassifier TestAccuracy: 0.9298245614035088
MLPClassifier TrainAccuracy: 0.9547738693467337
```

```
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 (Type-I):", FP1)
print("FN (Type-II):", FN1)
print("Type-I Error Rate in %:", type1_lr*100)
print("Type-II Error Rate in %:", type2_lr*100)
```

```

Logistic Regression Confusion Matrix:
[[ 64   4]
 [  3 100]]
FP (Type-I): 4
FN (Type-II): 3
Type-I Error Rate in %: 5.88235294117647
Type-II Error Rate in %: 2.912621359223301
```

```
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 (Type-I):", FP2)
print("FN (Type-II):", FN2)
print("Type-I Error Rate in %:", type1_dt*100)
print("Type-II Error Rate in %:", type2_dt*100)
```

```

Decision Tree Confusion Matrix:
[[62  6]
 [ 4 99]]
FP (Type-I): 6
FN (Type-II): 4
Type-I Error Rate in %: 8.823529411764707
Type-II Error Rate in %: 3.8834951456310676
```

```
cm_kn = confusion_matrix(y_test, pred_kn)
print("\n KNeigbors Confusion Matrix:")
print(cm_kn)
TN3, FP3, FN3, TP3 = cm_kn.ravel()
type1_kn = FP3 / (FP3 + TN3)
type2_kn = FN3 / (FN3 + TP3)

print("FP (Type-I):", FP3)
print("FN (Type-II):", FN3)
print("Type-I Error Rate in %:", type1_kn*100)
print("Type-II Error Rate in %:", type2_kn*100)
```

```
 KNeigbors Confusion Matrix:
[[ 57  11]
 [  3 100]]
FP (Type-I): 11
FN (Type-II): 3
Type-I Error Rate in %: 16.176470588235293
Type-II Error Rate in %: 2.912621359223301
```

```
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 (Type-I):", FP4)
print("FN (Type-II):", FN4)
print("Type-I Error Rate in %:", type1_nb*100)
print("Type-II Error Rate in %:", type2_nb*100)
```

```
Naive Bayes Confusion Matrix:
[[59  9]
 [ 6 97]]
FP (Type-I): 9
FN (Type-II): 6
Type-I Error Rate in %: 13.23529411764706
Type-II Error Rate in %: 5.825242718446602
```

```
cm_mlp = confusion_matrix(y_test, pred_mlp)
print("\nMLP Confusion Matrix:")
print(cm_mlp)
TN5, FP5, FN5, TP5 = cm_mlp.ravel()
type1_mlp = FP5 / (FP5 + TN5)
type2_mlp = FN5 / (FN5 + TP5)

print("FP (Type-I):", FP5)
print("FN (Type-II):", FN5)
print("Type-I Error Rate in %:", type1_mlp*100)
print("Type-II Error Rate in %:", type2_mlp*100)
```

```
MLP Confusion Matrix:
[[ 57  11]
 [  1 102]]
FP (Type-I): 11
FN (Type-II): 1
Type-I Error Rate in %: 16.176470588235293
Type-II Error Rate in %: 0.9708737864077669
```

```
train_acc_lr = accuracy_score(y_train, lr.predict(x_train))
test_acc_lr = accuracy_score(y_test, pred_lr)
precision_lr = precision_score(y_test, pred_lr)
recall_lr = recall_score(y_test, pred_lr)
f1_lr = f1_score(y_test, pred_lr)
roc_lr = roc_auc_score(y_test, pred_lr)

print("\n===== LOGISTIC REGRESSION RESULTS =====")
print("Training Accuracy :", train_acc_lr)
print("Test Accuracy     :", test_acc_lr)
print("Precision         :", precision_lr)
print("Recall            :", recall_lr)
print("F1-score          :", f1_lr)
print("ROC-AUC           :", roc_lr)
```

```
===== LOGISTIC REGRESSION RESULTS =====
Training Accuracy : 0.9547738693467337
Test Accuracy     : 0.9590643274853801
Precision         : 0.9615384615384616
Recall            : 0.970873786407767
F1-score          : 0.966183574879227
ROC-AUC           : 0.956025128498001
```

```
train_acc_dt = accuracy_score(y_train, dt.predict(x_train))
test_acc_dt = accuracy_score(y_test, pred_dt)
precision_dt = precision_score(y_test, pred_dt)
recall_dt = recall_score(y_test, pred_dt)
f1_dt = f1_score(y_test, pred_dt)
roc_dt = roc_auc_score(y_test, pred_dt)

print("\n===== DECISION TREE RESULTS =====")
print("Training Accuracy :", train_acc_dt)
print("Test Accuracy     :", test_acc_dt)
print("Precision         :", precision_dt)
print("Recall            :", recall_dt)
print("F1-score          :", f1_dt)
print("ROC-AUC           :", roc_dt)
```

```
===== DECISION TREE RESULTS =====
Training Accuracy : 1.0
Test Accuracy     : 0.9415204678362573
Precision         : 0.9428571428571428
Recall            : 0.9611650485436893
F1-score          : 0.9519230769230769
ROC-AUC           : 0.9364648772130211
```

```
pred_kn = kn.predict(x_test)
train_acc_kn = accuracy_score(y_train, kn.predict(x_train))
test_acc_kn = accuracy_score(y_test, pred_kn)
precision_kn = precision_score(y_test, pred_kn)
recall_kn = recall_score(y_test, pred_kn)
f1_kn = f1_score(y_test, pred_kn)
roc_kn = roc_auc_score(y_test, pred_kn)

print("\n===== KNN RESULTS =====")
print("Training Accuracy :", train_acc_kn)
print("Test Accuracy     :", test_acc_kn)
print("Precision         :", precision_kn)
print("Recall            :", recall_kn)
print("F1-score          :", f1_kn)
print("ROC-AUC           :", roc_kn)
```

```
===== KNN RESULTS =====
Training Accuracy : 0.9522613065326633
Test Accuracy     : 0.9181286549707602
Precision         : 0.9009009009009009
Recall            : 0.970873786407767
F1-score          : 0.9345794392523364
ROC-AUC           : 0.9045545402627071
```

```
pred_nb = nb.predict(x_test)
train_acc_nb = accuracy_score(y_train, nb.predict(x_train))
test_acc_nb = accuracy_score(y_test, pred_nb)
precision_nb = precision_score(y_test, pred_nb)
recall_nb = recall_score(y_test, pred_nb)
f1_nb = f1_score(y_test, pred_nb)
roc_nb = roc_auc_score(y_test, pred_nb)

print("\n===== NAIVE BAYES RESULTS =====")
print("Training Accuracy :", train_acc_nb)
print("Test Accuracy     :", test_acc_nb)
print("Precision         :", precision_nb)
print("Recall            :", recall_nb)
print("F1-score          :", f1_nb)
print("ROC-AUC           :", roc_nb)
```

```
===== NAIVE BAYES RESULTS =====
Training Accuracy : 0.949748743718593
```

```
Test Accuracy     : 0.9122807017543859
Precision         : 0.9150943396226415
Recall            : 0.941747572815534
F1-score          : 0.9282296650717703
ROC-AUC           : 0.9046973158195317
```

```
pred_mlp = mlp.predict(x_test)
train_acc_mlp = accuracy_score(y_train, mlp.predict(x_train))
test_acc_mlp = accuracy_score(y_test, pred_mlp)
precision_mlp = precision_score(y_test, pred_mlp)
recall_mlp = recall_score(y_test, pred_mlp)
f1_mlp = f1_score(y_test, pred_mlp)
```