neural_net

January 20, 2022

[ ]:
```
#Core libraries
import pandas as pd
import numpy as np


#Preprocessing & essentials
from sklearn.model_selection import train_test_split
import seaborn as sns
import matplotlib.pyplot as plt

#Neural Network Algorithms
import keras
import tensorflow as tf
from keras import backend as K
from keras.models import Sequential
from keras.layers import Activation
from keras.layers.core import Dense

#Machine Learning Algorithms
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB

#performance Metrics
from sklearn.metrics import classification_report, accuracy_score
```

[ ]:
```
#importing and reading dataset
data = pd.read_csv('birth_type.csv')
data.head()
```

[ ]:

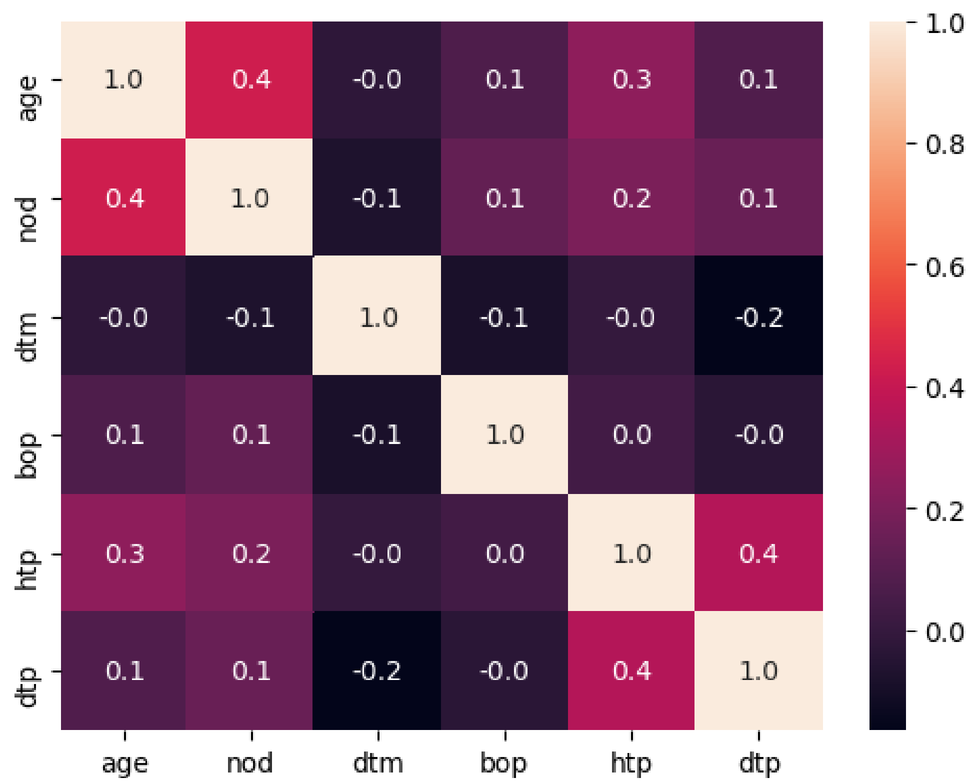
	age	nod	dtm	bop	htp	dtp
0	22	1	0	2	0	0
1	26	2	0	1	0	1
2	26	2	1	1	0	0
3	28	1	0	2	0	0
4	22	2	0	1	0	1

[ ]:
```
#--------Data Description-----------
#age = Mother's Age
#nod = Number of Previous Delivery (1 - 4) Delivery Times
#dtm = Delivery Time (0 = Timely, 1 = Premature, 2 = Latecomer)
#bop = Blood of Pressure (0 = Low, 1 = Normal, 2 = High)
#htp = Heart Proble (0 = Apt, 1=Inept)
#dtp = Delivery Type (0 = Spontaneous Vaginal Delivery, 1 = Cesarean Section)

data.describe()
```

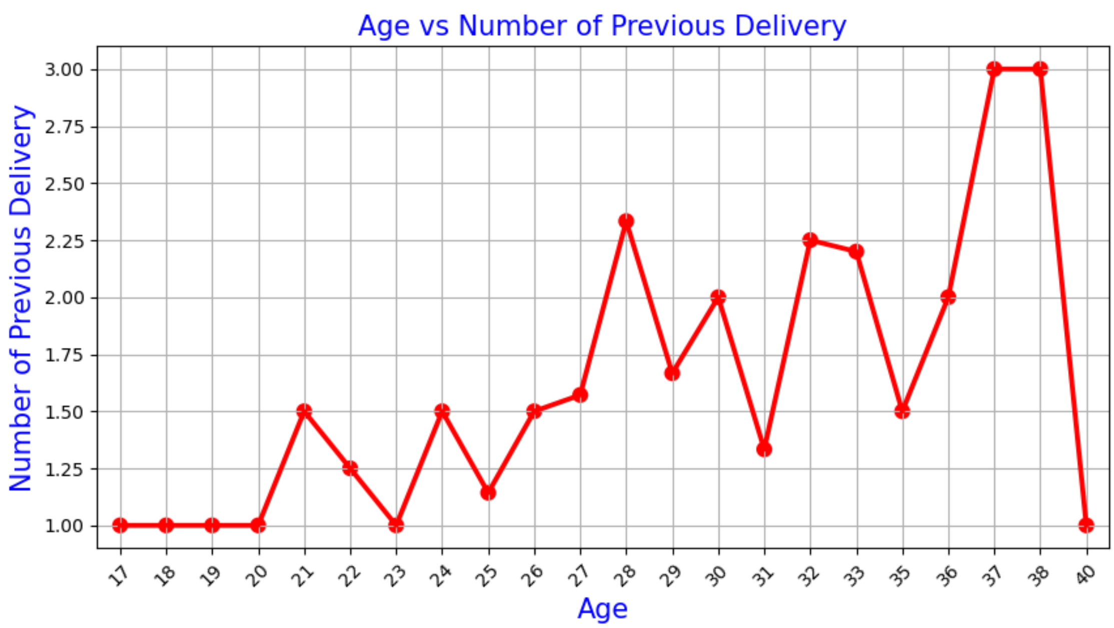
[ ]:

	age	nod	dtm	bop	htp	dtp
count	80.000000	80.000000	80.000000	80.000000	80.000000	80.000000
mean	27.687500	1.662500	0.637500	1.000000	0.375000	0.575000
std	5.017927	0.794662	0.815107	0.711568	0.487177	0.497462
min	17.000000	1.000000	0.000000	0.000000	0.000000	0.000000
25%	25.000000	1.000000	0.000000	0.750000	0.000000	0.000000
50%	27.000000	1.000000	0.000000	1.000000	0.000000	1.000000
75%	32.000000	2.000000	1.000000	1.250000	1.000000	1.000000
max	40.000000	4.000000	2.000000	2.000000	1.000000	1.000000

[ ]:
```
#Data Correlation

sns.heatmap(data.corr(),annot=True,fmt='.1f')
plt.show()
```



[ ]:
```
#ploting Age vs Number of Previous Delivery

age_unique=sorted(data.age.unique())
age_nod_values=data.groupby('age')['nod'].count().values
mean_nod=[]
for i,age in enumerate(age_unique):
    mean_nod.append(sum(data[data['age']==age].nod)/age_nod_values[i])

plt.figure(figsize=(10,5))
sns.pointplot(x=age_unique,y=mean_nod,color='red',alpha=0.8)
plt.xlabel('Age',fontsize = 15,color='blue')
plt.xticks(rotation=45)
plt.ylabel('Number of Previous Delivery',fontsize = 15,color='blue')
plt.title('Age vs Number of Previous Delivery',fontsize = 15,color='blue')
plt.grid()
plt.show()
```



[ ]:
```
#Dividing Dependent & Independent Variables

X = data[['age', 'nod', 'dtm', 'htp']]
y = data['dtp']
```

[ ]:
```
#Dividing Dataset into testing and training dataset
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.8,␣
 ↪test_size=0.2, random_state=42)
```

[ ]:
```
#Neural Network

model = Sequential([
    Dense(16, input_shape=(4,), activation='relu'),
    Dense(32, activation='relu'),
    Dense(2, activation='sigmoid'),
])
```

[ ]:
```
#Supervised Learning Condition

adam = tf.keras.optimizers.Adam(learning_rate=0.0001, decay=1e-6)
model.compile(optimizer='rmsprop', loss='sparse_categorical_crossentropy',␣
 ↪metrics=['accuracy'])
```

[ ]:
```
#Model Training

history = model.fit(x=X_train, y=y_train, validation_data=(X_test, y_test),
                    batch_size=5, epochs=50, shuffle=True, verbose=2)
```

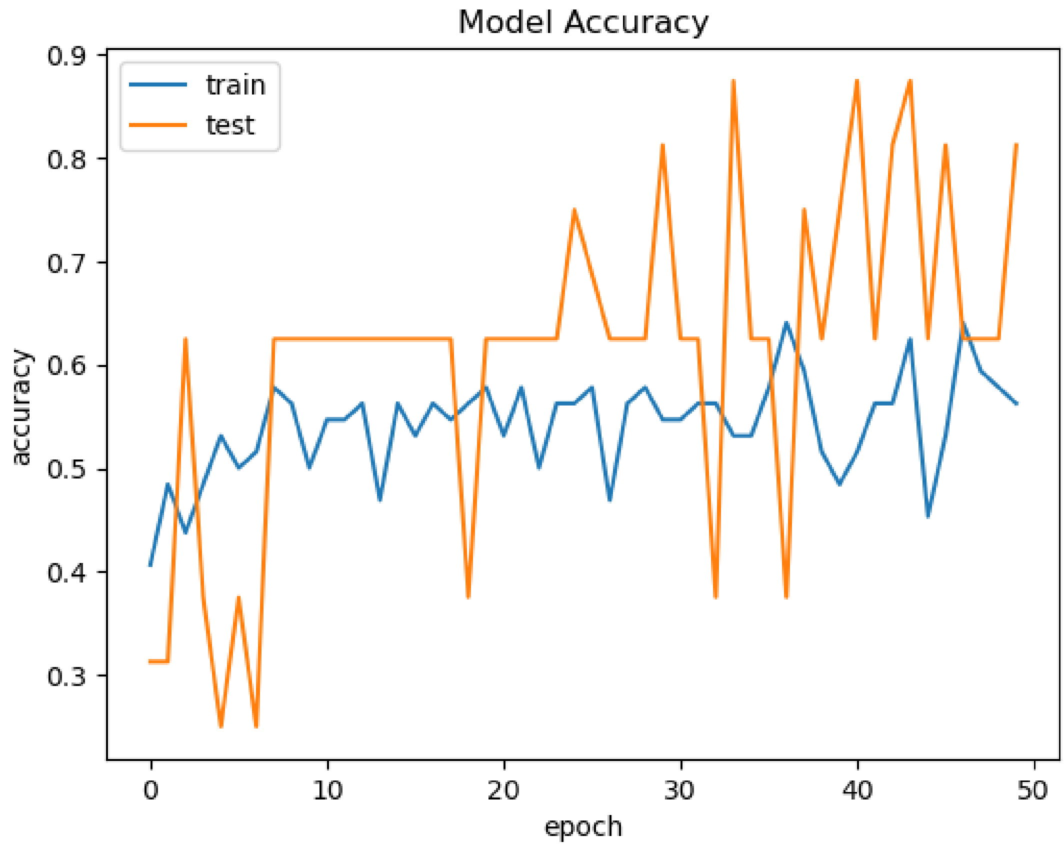
```
Epoch 1/50
13/13 - 21s - loss: 0.9982 - accuracy: 0.4062 - val_loss: 0.7094 - val_accuracy:
0.3125
Epoch 2/50
13/13 - 0s - loss: 0.7165 - accuracy: 0.4844 - val_loss: 0.7085 - val_accuracy:
0.3125
Epoch 3/50
13/13 - 0s - loss: 0.7201 - accuracy: 0.4375 - val_loss: 0.6923 - val_accuracy:
0.6250
Epoch 4/50
13/13 - 0s - loss: 0.7188 - accuracy: 0.4844 - val_loss: 0.7294 - val_accuracy:
0.3750
Epoch 5/50
13/13 - 0s - loss: 0.7162 - accuracy: 0.5312 - val_loss: 0.7122 - val_accuracy:
0.2500
Epoch 6/50
13/13 - 0s - loss: 0.7103 - accuracy: 0.5000 - val_loss: 0.7578 - val_accuracy:
0.3750
Epoch 7/50
13/13 - 0s - loss: 0.7106 - accuracy: 0.5156 - val_loss: 0.7003 - val_accuracy:
0.2500
Epoch 8/50
13/13 - 0s - loss: 0.7162 - accuracy: 0.5781 - val_loss: 0.6610 - val_accuracy:
0.6250
Epoch 9/50
```

```
13/13 - 0s - loss: 0.7136 - accuracy: 0.5625 - val_loss: 0.6781 - val_accuracy:
0.6250
Epoch 10/50
13/13 - 0s - loss: 0.7047 - accuracy: 0.5000 - val_loss: 0.6636 - val_accuracy:
0.6250
Epoch 11/50
13/13 - 0s - loss: 0.7042 - accuracy: 0.5469 - val_loss: 0.6838 - val_accuracy:
0.6250
Epoch 12/50
13/13 - 0s - loss: 0.6989 - accuracy: 0.5469 - val_loss: 0.6602 - val_accuracy:
0.6250
Epoch 13/50
13/13 - 0s - loss: 0.7064 - accuracy: 0.5625 - val_loss: 0.6715 - val_accuracy:
0.6250
Epoch 14/50
13/13 - 0s - loss: 0.7113 - accuracy: 0.4688 - val_loss: 0.6614 - val_accuracy:
0.6250
Epoch 15/50
13/13 - 0s - loss: 0.7036 - accuracy: 0.5625 - val_loss: 0.6865 - val_accuracy:
0.6250
Epoch 16/50
13/13 - 0s - loss: 0.6937 - accuracy: 0.5312 - val_loss: 0.6526 - val_accuracy:
0.6250
Epoch 17/50
13/13 - 0s - loss: 0.7181 - accuracy: 0.5625 - val_loss: 0.6770 - val_accuracy:
0.6250
Epoch 18/50
13/13 - 0s - loss: 0.6917 - accuracy: 0.5469 - val_loss: 0.6759 - val_accuracy:
0.6250
Epoch 19/50
13/13 - 0s - loss: 0.6915 - accuracy: 0.5625 - val_loss: 0.6970 - val_accuracy:
0.3750
Epoch 20/50
13/13 - 0s - loss: 0.7107 - accuracy: 0.5781 - val_loss: 0.6695 - val_accuracy:
0.6250
Epoch 21/50
13/13 - 0s - loss: 0.6850 - accuracy: 0.5312 - val_loss: 0.6475 - val_accuracy:
0.6250
Epoch 22/50
13/13 - 0s - loss: 0.7068 - accuracy: 0.5781 - val_loss: 0.6466 - val_accuracy:
0.6250
Epoch 23/50
13/13 - 0s - loss: 0.6908 - accuracy: 0.5000 - val_loss: 0.6414 - val_accuracy:
0.6250
Epoch 24/50
13/13 - 0s - loss: 0.6869 - accuracy: 0.5625 - val_loss: 0.6479 - val_accuracy:
0.6250
Epoch 25/50
```

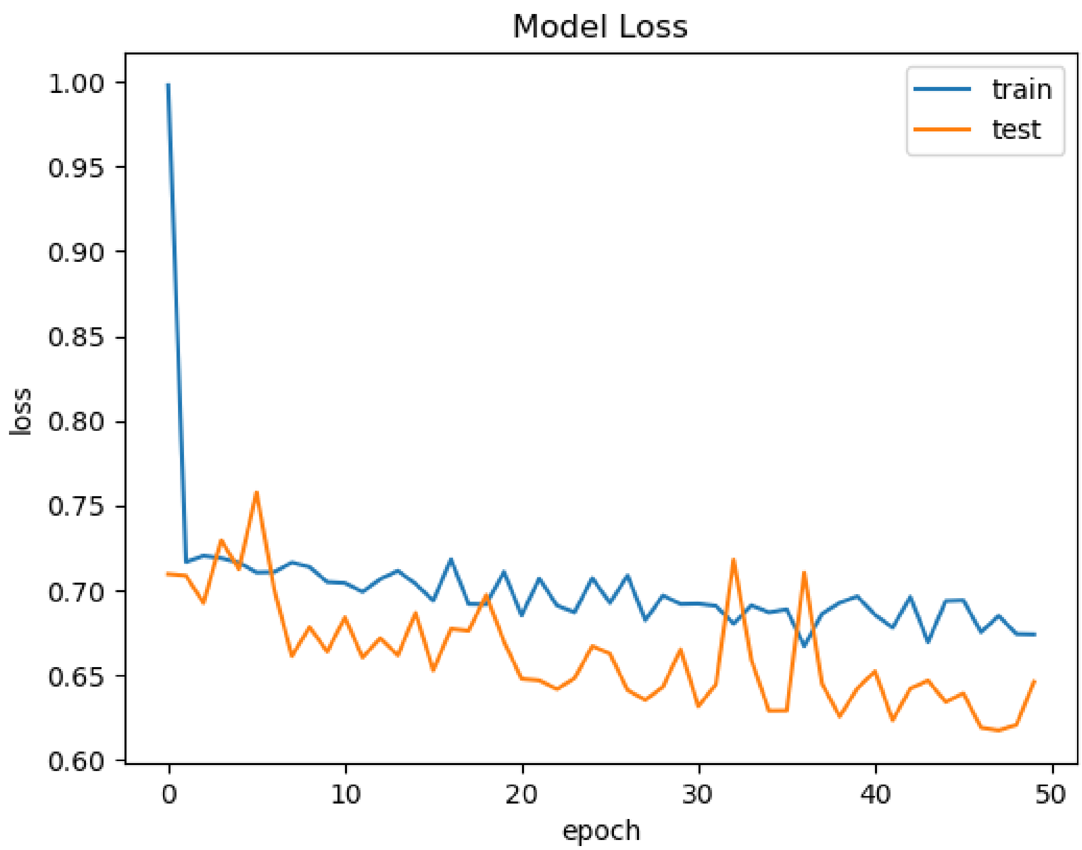
```
13/13 - 0s - loss: 0.7070 - accuracy: 0.5625 - val_loss: 0.6668 - val_accuracy:
0.7500
Epoch 26/50
13/13 - 0s - loss: 0.6923 - accuracy: 0.5781 - val_loss: 0.6624 - val_accuracy:
0.6875
Epoch 27/50
13/13 - 0s - loss: 0.7086 - accuracy: 0.4688 - val_loss: 0.6408 - val_accuracy:
0.6250
Epoch 28/50
13/13 - 0s - loss: 0.6821 - accuracy: 0.5625 - val_loss: 0.6350 - val_accuracy:
0.6250
Epoch 29/50
13/13 - 0s - loss: 0.6967 - accuracy: 0.5781 - val_loss: 0.6428 - val_accuracy:
0.6250
Epoch 30/50
13/13 - 0s - loss: 0.6917 - accuracy: 0.5469 - val_loss: 0.6650 - val_accuracy:
0.8125
Epoch 31/50
13/13 - 0s - loss: 0.6919 - accuracy: 0.5469 - val_loss: 0.6313 - val_accuracy:
0.6250
Epoch 32/50
13/13 - 0s - loss: 0.6906 - accuracy: 0.5625 - val_loss: 0.6443 - val_accuracy:
0.6250
Epoch 33/50
13/13 - 0s - loss: 0.6801 - accuracy: 0.5625 - val_loss: 0.7179 - val_accuracy:
0.3750
Epoch 34/50
13/13 - 0s - loss: 0.6909 - accuracy: 0.5312 - val_loss: 0.6588 - val_accuracy:
0.8750
Epoch 35/50
13/13 - 0s - loss: 0.6868 - accuracy: 0.5312 - val_loss: 0.6287 - val_accuracy:
0.6250
Epoch 36/50
13/13 - 0s - loss: 0.6885 - accuracy: 0.5781 - val_loss: 0.6288 - val_accuracy:
0.6250
Epoch 37/50
13/13 - 0s - loss: 0.6668 - accuracy: 0.6406 - val_loss: 0.7102 - val_accuracy:
0.3750
Epoch 38/50
13/13 - 0s - loss: 0.6860 - accuracy: 0.5938 - val_loss: 0.6446 - val_accuracy:
0.7500
Epoch 39/50
13/13 - 0s - loss: 0.6924 - accuracy: 0.5156 - val_loss: 0.6252 - val_accuracy:
0.6250
Epoch 40/50
13/13 - 0s - loss: 0.6962 - accuracy: 0.4844 - val_loss: 0.6418 - val_accuracy:
0.7500
Epoch 41/50
```

```
13/13 - 0s - loss: 0.6854 - accuracy: 0.5156 - val_loss: 0.6521 - val_accuracy:
0.8750
Epoch 42/50
13/13 - 0s - loss: 0.6778 - accuracy: 0.5625 - val_loss: 0.6231 - val_accuracy:
0.6250
Epoch 43/50
13/13 - 0s - loss: 0.6960 - accuracy: 0.5625 - val_loss: 0.6418 - val_accuracy:
0.8125
Epoch 44/50
13/13 - 0s - loss: 0.6692 - accuracy: 0.6250 - val_loss: 0.6465 - val_accuracy:
0.8750
Epoch 45/50
13/13 - 0s - loss: 0.6935 - accuracy: 0.4531 - val_loss: 0.6338 - val_accuracy:
0.6250
Epoch 46/50
13/13 - 0s - loss: 0.6939 - accuracy: 0.5312 - val_loss: 0.6389 - val_accuracy:
0.8125
Epoch 47/50
13/13 - 0s - loss: 0.6750 - accuracy: 0.6406 - val_loss: 0.6187 - val_accuracy:
0.6250
Epoch 48/50
13/13 - 0s - loss: 0.6847 - accuracy: 0.5938 - val_loss: 0.6171 - val_accuracy:
0.6250
Epoch 49/50
13/13 - 0s - loss: 0.6740 - accuracy: 0.5781 - val_loss: 0.6203 - val_accuracy:
0.6250
Epoch 50/50
13/13 - 0s - loss: 0.6738 - accuracy: 0.5625 - val_loss: 0.6457 - val_accuracy:
0.8125
```

[ ]:
```python
#Training Model Accuracy

plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'])
plt.show()
```



[ ]:
```
#Model Value Loss

plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'])
plt.show()
```



[ ]:
```
#Machine Learning with Random Forest & Naive Bayes
rfg = RandomForestClassifier(max_depth=2, random_state=42)
nby = GaussianNB()

#Training Process
rfg.fit(X_train, y_train)
nby.fit(X_train, y_train)
```

[ ]:
```
GaussianNB()
```

[ ]:
```
#Performance Evaluation with Prediction Score & Classification Report
y_snn = np.argmax(model.predict(X_test), axis=1)
y_rfg = rfg.predict(X_test)
y_nby = nby.predict(X_test)

nn_scr = accuracy_score(y_test, y_snn)
rfg_scr = rfg.score(X_test, y_test)
nby_scr = nby.score(X_test, y_test)
```

```
[ ]: #Accuracy Score Output
print('Prediction Accuracy Score for Neural Network = ', nn_scr*100,'%')
print('Prediction Accuracy Score for Random Forest = ', rfg_scr*100,'%')
print('Prediction Accuracy Score for Naive Bayes = ', nby_scr*100,'%')
```

```
Prediction Accuracy Score for Neural Network =  81.25 %
Prediction Accuracy Score for Random Forest =  68.75 %
Prediction Accuracy Score for Naive Bayes =  75.0 %
```

```
[ ]: #Classification Report Output
print("Report for Neural Network")
print(classification_report(y_test, y_snn))
print("")
print("Report for Random Forest")
print(classification_report(y_test, y_rfg))
print("")
print("Report for Naive Bayes")
print(classification_report(y_test, y_nby))
```

```
Report for Neural Network
              precision    recall  f1-score   support

           0       0.71      0.83      0.77         6
           1       0.89      0.80      0.84        10

    accuracy                           0.81        16
   macro avg       0.80      0.82      0.81        16
weighted avg       0.82      0.81      0.81        16


Report for Random Forest
              precision    recall  f1-score   support

           0       0.60      0.50      0.55         6
           1       0.73      0.80      0.76        10

    accuracy                           0.69        16
   macro avg       0.66      0.65      0.65        16
weighted avg       0.68      0.69      0.68        16


Report for Naive Bayes
              precision    recall  f1-score   support

           0       0.67      0.67      0.67         6
           1       0.80      0.80      0.80        10

    accuracy                           0.75        16
```

```
   macro avg       0.73      0.73      0.73        16
weighted avg       0.75      0.75      0.75        16
```