Name: Raven Jacinto Course and Section: CPE019 - CPE32S3 Date of Submission: Jan. 31, 2024 Instructor: Engr. Roman Richard Activity: Assignment 5.2: Build and Apply Multilayer Perceptron

In this assignment, you are task to build a multilayer perceptron model. The following are the requirements:

-Choose any dataset

-Explain the problem you are trying to solve

-Create your own model

-Evaluate the accuracy of your model

Problem Explanation:

The dataset I have is hospital records which contains all the laboratory results of 303 patients that is having a laboratory checkup related to heart condition and status.

The problem I want to solve is predicting if the patient is has aquired heart disease or not based on the results of their laboratory checkups.

Importing needed libraries, packages, etc.

```
pip install pandas numpy scikit-learn tensorflow matplotlib seaborn
scipy keras

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, confusion_matrix,
classification_report
import seaborn as sns
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
```

Loading the Dataset

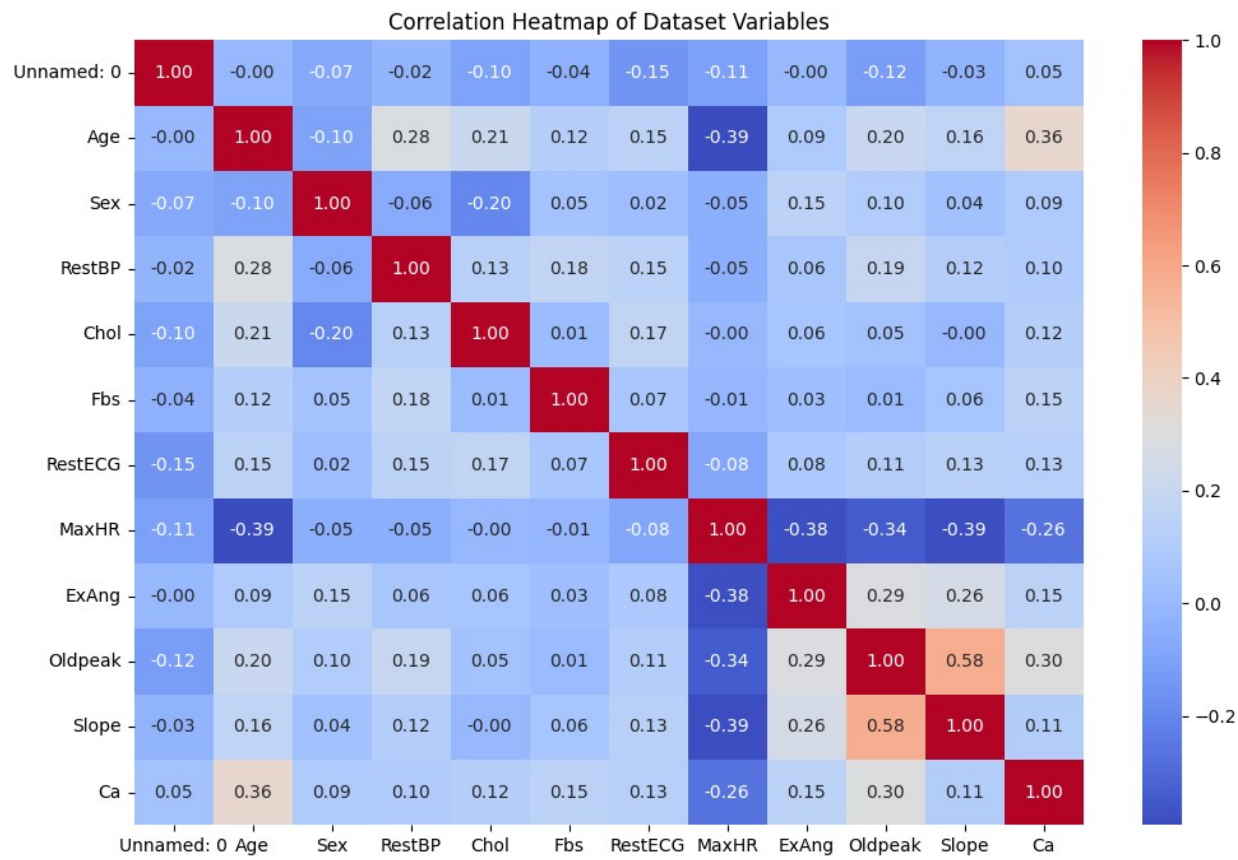
```
# Loading Dataset
df = pd.read_csv("/content/HeartDiseaseData.csv")
```

Preprocessing the Dataset

```
# Calculate the correlation matrix
correlation_matrix = df.corr()
```

```
# Plot the correlation matrix using Seaborn's heatmap
plt.figure(figsize=(12, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm',
fmt=".2f")
plt.title('Correlation Heatmap of Dataset Variables')
plt.show()
```

```
<ipython-input-471-61d08e138547>:2: FutureWarning: The default value
of numeric_only in DataFrame.corr is deprecated. In a future version,
it will default to False. Select only valid columns or specify the
value of numeric_only to silence this warning.
  correlation_matrix = df.corr()
```



Observation: The correlation values are far or low for each features. Therefore, it is a good dataset to use for multilayer perceptron model.

```
print(df.shape)
```

```
(303, 15)
```

Observation: I have 303 rows and 15 columns or features.

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 303 entries, 0 to 302
Data columns (total 15 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Unnamed: 0  303 non-null    int64
 1   Age         303 non-null    int64
 2   Sex         303 non-null    int64
 3   ChestPain   303 non-null    object
 4   RestBP      303 non-null    int64
 5   Chol        303 non-null    int64
 6   Fbs         303 non-null    int64
 7   RestECG     303 non-null    int64
 8   MaxHR       303 non-null    int64
 9   ExAng       303 non-null    int64
 10  Oldpeak     303 non-null    float64
 11  Slope       303 non-null    int64
 12  Ca          299 non-null    float64
 13  Thal        301 non-null    object
 14  AHD         303 non-null    object
dtypes: float64(2), int64(10), object(3)
memory usage: 35.6+ KB
```

Observation: Found out that there are null values in Ca and Thal columns.

```
df.isnull().sum()
```

```
Unnamed: 0    0
Age           0
Sex           0
ChestPain     0
RestBP        0
Chol          0
Fbs           0
RestECG       0
MaxHR         0
ExAng         0
Oldpeak       0
Slope         0
Ca            4
Thal          2
AHD           0
dtype: int64
```

Observation: We got 6 total null values in Ca and Thal column, since we have null values, we are going to drop it

```
df = df.dropna()
```

```
df = df.drop(columns = 'Unnamed: 0')
```

Observation: Noticed the Unnamed: 0 column which is not important to this dataset, so I dropped the entire column.

```
df.isnull().sum()

Age          0
Sex          0
ChestPain    0
RestBP       0
Chol         0
Fbs          0
RestECG      0
MaxHR        0
ExAng        0
Oldpeak      0
Slope        0
Ca           0
Thal         0
AHD          0
dtype: int64
```

Observation: The data does not have any null values.

```
df.tail()
```

```
[478] df.tail()
```

	Age	Sex	ChestPain	RestBP	Chol	Fbs	RestECG	MaxHR	ExAng	Oldpeak	Slope	Ca	Thal	AHD
297	57	0	asymptomatic	140	241	0	0	123	1	0.2	2	0.0	reversable	Yes
298	45	1	typical	110	264	0	0	132	0	1.2	2	0.0	reversable	Yes
299	68	1	asymptomatic	144	193	1	0	141	0	3.4	2	2.0	reversable	Yes
300	57	1	asymptomatic	130	131	0	0	115	1	1.2	2	1.0	reversable	Yes
301	57	0	nontypical	130	236	0	2	174	0	0.0	2	1.0	normal	Yes

Observation: However, the data has categorical values which cannot be processed in MLP, so I need to convert in into numerical values.

Feature selection/engineering

```
df['AHD'] = df['AHD'].astype('category')
df['AHD'] = df['AHD'].cat.codes

df['ChestPain'] = df['ChestPain'].astype('category')
df['ChestPain'] = df['ChestPain'].cat.codes

df['Thal'] = df['Thal'].astype('category')
df['Thal'] = df['Thal'].cat.codes
```

```
df
```

	Age	Sex	ChestPain	RestBP	Chol	Fbs	RestECG	MaxHR	ExAng	Oldpeak	Slope	Ca	Thal	AHD
0	63	1	3	145	233	1	2	150	0	2.3	3	0.0	0	0
1	67	1	0	160	286	0	2	108	1	1.5	2	3.0	1	1
2	67	1	0	120	229	0	2	129	1	2.6	2	2.0	2	1
3	37	1	1	130	250	0	0	187	0	3.5	3	0.0	1	0
4	41	0	2	130	204	0	2	172	0	1.4	1	0.0	1	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
297	57	0	0	140	241	0	0	123	1	0.2	2	0.0	2	1
298	45	1	3	110	264	0	0	132	0	1.2	2	0.0	2	1
299	68	1	0	144	193	1	0	141	0	3.4	2	2.0	2	1
300	57	1	0	130	131	0	0	115	1	1.2	2	1.0	2	1
301	57	0	2	130	236	0	2	174	0	0.0	2	1.0	1	1

297 rows × 14 columns

Observation: As we can see, every data in columns and rows are now in number forms which means that it is ready for the Multilayer Perceptron Model.

```
# Splitting the independent variable X to dependent variable or target
value (y)
X = df.drop(columns = 'AHD')
y = df['AHD']


# We split the data because the test set must be separated from the
training set
# to provide an unbiased estimate of the model's performance.

# Using this splitting method, it trains the model on one set and
evaluate its performance on another set
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size =
0.2, random_state = 42)

#checking the values of our split data
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(237, 13)
(60, 13)
```

```
(237,)
(60,)
```

Implementing MLP using Keras

```
model = tf.keras.Sequential([
    tf.keras.layers.Dense(32, activation='relu',
input_shape=(X_train.shape[1],)),
    tf.keras.layers.Dense(16, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
```

Observation: Relu to allow model to learn complex paterns for non linear data while Sigmoid is used on the last layer to make the output 0's and 1's since it fits to my dataset based on checking.

```
# Compiling the model

model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])
```

Observation: I used adam optimizer for gradient-based optimization algorithms, then used binary_crossentropy as loss for binary classification problems, and use accuracy as metrics to measure the proportion of correct predictions made by the model.

Model Training

```
#training the model
model.fit(X_train, y_train, epochs=50)
#result of the training model

Epoch 1/50
8/8 [==============================] - 0s 4ms/step - loss: 0.4090 -
accuracy: 0.8101
Epoch 2/50
8/8 [==============================] - 0s 5ms/step - loss: 0.4144 -
accuracy: 0.7975
Epoch 3/50
8/8 [==============================] - 0s 5ms/step - loss: 0.4091 -
accuracy: 0.8481
Epoch 4/50
8/8 [==============================] - 0s 5ms/step - loss: 0.4033 -
accuracy: 0.8270
Epoch 5/50
8/8 [==============================] - 0s 8ms/step - loss: 0.3791 -
accuracy: 0.8397
Epoch 6/50
8/8 [==============================] - 0s 9ms/step - loss: 0.3899 -
```

```
accuracy: 0.8312
Epoch 7/50
8/8 [==============================] - 0s 9ms/step - loss: 0.3996 -
accuracy: 0.8354
Epoch 8/50
8/8 [==============================] - 0s 11ms/step - loss: 0.4249 -
accuracy: 0.8312
Epoch 9/50
8/8 [==============================] - 0s 7ms/step - loss: 0.4140 -
accuracy: 0.8143
Epoch 10/50
8/8 [==============================] - 0s 6ms/step - loss: 0.4158 -
accuracy: 0.8101
Epoch 11/50
8/8 [==============================] - 0s 6ms/step - loss: 0.4373 -
accuracy: 0.8017
Epoch 12/50
8/8 [==============================] - 0s 6ms/step - loss: 0.3809 -
accuracy: 0.8397
Epoch 13/50
8/8 [==============================] - 0s 5ms/step - loss: 0.3886 -
accuracy: 0.8354
Epoch 14/50
8/8 [==============================] - 0s 5ms/step - loss: 0.4173 -
accuracy: 0.8017
Epoch 15/50
8/8 [==============================] - 0s 6ms/step - loss: 0.3750 -
accuracy: 0.8481
Epoch 16/50
8/8 [==============================] - 0s 5ms/step - loss: 0.3755 -
accuracy: 0.8481
Epoch 17/50
8/8 [==============================] - 0s 6ms/step - loss: 0.3852 -
accuracy: 0.8270
Epoch 18/50
8/8 [==============================] - 0s 7ms/step - loss: 0.4105 -
accuracy: 0.8270
Epoch 19/50
8/8 [==============================] - 0s 5ms/step - loss: 0.3730 -
accuracy: 0.8523
Epoch 20/50
8/8 [==============================] - 0s 6ms/step - loss: 0.3938 -
accuracy: 0.8523
Epoch 21/50
8/8 [==============================] - 0s 6ms/step - loss: 0.3928 -
accuracy: 0.8270
Epoch 22/50
8/8 [==============================] - 0s 5ms/step - loss: 0.4449 -
accuracy: 0.8101
```

```
Epoch 23/50
8/8 [==============================] - 0s 6ms/step - loss: 0.3938 -
accuracy: 0.8439
Epoch 24/50
8/8 [==============================] - 0s 7ms/step - loss: 0.4007 -
accuracy: 0.8228
Epoch 25/50
8/8 [==============================] - 0s 6ms/step - loss: 0.4141 -
accuracy: 0.8186
Epoch 26/50
8/8 [==============================] - 0s 6ms/step - loss: 0.4245 -
accuracy: 0.8312
Epoch 27/50
8/8 [==============================] - 0s 7ms/step - loss: 0.3966 -
accuracy: 0.8354
Epoch 28/50
8/8 [==============================] - 0s 6ms/step - loss: 0.3722 -
accuracy: 0.8439
Epoch 29/50
8/8 [==============================] - 0s 7ms/step - loss: 0.3775 -
accuracy: 0.8439
Epoch 30/50
8/8 [==============================] - 0s 5ms/step - loss: 0.4281 -
accuracy: 0.8228
Epoch 31/50
8/8 [==============================] - 0s 4ms/step - loss: 0.3952 -
accuracy: 0.8228
Epoch 32/50
8/8 [==============================] - 0s 4ms/step - loss: 0.3788 -
accuracy: 0.8312
Epoch 33/50
8/8 [==============================] - 0s 4ms/step - loss: 0.3758 -
accuracy: 0.8270
Epoch 34/50
8/8 [==============================] - 0s 4ms/step - loss: 0.3877 -
accuracy: 0.8397
Epoch 35/50
8/8 [==============================] - 0s 4ms/step - loss: 0.3694 -
accuracy: 0.8397
Epoch 36/50
8/8 [==============================] - 0s 4ms/step - loss: 0.3680 -
accuracy: 0.8565
Epoch 37/50
8/8 [==============================] - 0s 3ms/step - loss: 0.3677 -
accuracy: 0.8439
Epoch 38/50
8/8 [==============================] - 0s 4ms/step - loss: 0.3871 -
accuracy: 0.8354
Epoch 39/50
```

```
8/8 [==============================] - 0s 4ms/step - loss: 0.3686 -
accuracy: 0.8439
Epoch 40/50
8/8 [==============================] - 0s 4ms/step - loss: 0.3677 -
accuracy: 0.8565
Epoch 41/50
8/8 [==============================] - 0s 4ms/step - loss: 0.3785 -
accuracy: 0.8397
Epoch 42/50
8/8 [==============================] - 0s 4ms/step - loss: 0.4101 -
accuracy: 0.8354
Epoch 43/50
8/8 [==============================] - 0s 4ms/step - loss: 0.4182 -
accuracy: 0.7975
Epoch 44/50
8/8 [==============================] - 0s 4ms/step - loss: 0.4360 -
accuracy: 0.8101
Epoch 45/50
8/8 [==============================] - 0s 3ms/step - loss: 0.4924 -
accuracy: 0.7764
Epoch 46/50
8/8 [==============================] - 0s 3ms/step - loss: 0.4515 -
accuracy: 0.8186
Epoch 47/50
8/8 [==============================] - 0s 2ms/step - loss: 0.4091 -
accuracy: 0.8186
Epoch 48/50
8/8 [==============================] - 0s 3ms/step - loss: 0.3742 -
accuracy: 0.8481
Epoch 49/50
8/8 [==============================] - 0s 3ms/step - loss: 0.3683 -
accuracy: 0.8439
Epoch 50/50
8/8 [==============================] - 0s 3ms/step - loss: 0.3641 -
accuracy: 0.8397

<keras.src.callbacks.History at 0x7b8c97f453f0>
```

Final Model Evaluation

```
#evaluation of training set
model.evaluate(X_train, y_train)

8/8 [==============================] - 0s 3ms/step - loss: 0.3625 -
accuracy: 0.8439

[0.362544983625412, 0.8438818454742432]
```

Observation: My training set gives 84% accuracy with 36% loss.

```
#Evaluation of testing set
model.evaluate(X_test, y_test)
#Actual accuracy of the model to the Xtest and Ytest

2/2 [==============================] - 0s 10ms/step - loss: 0.3486 - 
accuracy: 0.8167

[0.34857144951820374, 0.8166666626930237]
```

Observation: My test set gives 81% accuracy with 34% loss.

Evaluation results, prove that the splitting of data is good. It is also close to each other, meaning MLP model can be trusted since the test set gives 81% accuracy.

```
# Sample input for prediction
sample_input = [67,1,0,160,286,0,2,108,1,1.5,2,3.0,1]
AHDprediction = model.predict([sample_input])[0]

# Checking if the probability is 1 or 0
print('\n', 'Sample Laboratory Result: ', sample_input, '\n')

if AHDprediction >= 0.99:
    print('Prediction: The person will most likely aquired heart 
disease')
else:
    print('Prediction: The person will most likely NOT aquired heart 
disease')

1/1 [==============================] - 0s 142ms/step

 Sample Laboratory Result:  [67, 1, 0, 160, 286, 0, 2, 108, 1, 1.5, 2, 
3.0, 1]

Prediction: The person will most likely aquired heart disease
```

Conclusion:

In conclusion, I solved the problem in predicting the possibility of heart disease in patients based on their laboratory results using the multilayer perceptron model. With the help of internet resources like youtube and google, I managed to complete this activity.

Summary:

In summary, I pre-processed the dataset I used for MLP model and split the data into 80% training and 20% testing to suffice the training with enough data and understand some complexity better. After thorough final checking of the data, I affirmed that it is ready for the training. So I assembled the MLP with the help of Keras, set the neurons for each layer and choose proper activation for my dataset, considering 50 iterations or epoch = 50 for the model to learn multiple times from the data. Then I proceed to evaluation of model training and testing set. Which I got 84% accuracy for training and 81% accuracy for testing. Lastly, I created an if else statement and sample input to prove that the model can really predict the desired output using the multilayer perceptron model.