BUENAFE, Lorenz Angelo N.
1915058
CPE 019 - CPE32S9

Choose any dataset applicable to the classification problem, and also, choose any dataset applicable to the regression problem.

Classification - https://www.kaggle.com/datasets/bharath011/heart-disease-classification-dataset

Regression - https://www.kaggle.com/datasets/anubhavgoyal10/laptop-prices-dataset

Explain your datasets and the problem being addressed.

https://www.kaggle.com/datasets/bharath011/heart-disease-classification-dataset
The main purpose here is to collect characteristics of Heart Attack or factors that contribute to it.

https://www.kaggle.com/datasets/anubhavgoyal10/laptop-prices-dataset
This dataset can be used for regression analysis to predict the prices of laptops based on their features.

For classification, do the following:
Create a base model
Evaluate the model with k-fold cross validation
Improve the accuracy of your model by applying additional hidden layers

1. Creating a base model

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
from sklearn.model_selection import cross_val_score
```

```
df = pd.read_csv('/content/drive/MyDrive/CPE 019 Emerging Technologies 3/Assignment 7.1/Hea
```

```
print(df.head)
```

```
<bound method NDFrame.head of       age  gender  impluse  pressurehight  pressurelow  g
0      64       1       66            160           83   160.0   1.80
1      21       1       94             98           46   296.0   6.75
2      55       1       64            160           77   270.0   1.99
3      64       1       70            120           55   270.0  13.87
4      55       1       64            112           65   300.0   1.08
...   ...     ...      ...            ...          ...     ...    ...
1314   44       1       94            122           67   204.0   1.63
1315   66       1       84            125           55   149.0   1.33
1316   45       1       85            168          104    96.0   1.24
1317   54       1       58            117           68   443.0   5.80
1318   51       1       94            157           79   134.0  50.89

      troponin     class
0        0.012  negative
1        1.060  positive
```

```
2        0.003  negative
3        0.122  positive
4        0.003  negative
...        ...       ...
1314     0.006  negative
1315     0.172  positive
1316     4.250  positive
1317     0.359  positive
1318     1.770  positive

[1319 rows x 9 columns]>
```

```
X = df.iloc[:, 0:4].values
y = df.iloc[:, 4].values
```

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

```
# Feature scaling
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

```
mlp_classifier = MLPClassifier(hidden_layer_sizes=(50,), activation='relu', alpha=0.0001, b
mlp_classifier.fit(X_train, y_train)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/neural_network/_multilayer_perceptron.p
  warnings.warn(
▾                         MLPClassifier
MLPClassifier(batch_size=256, hidden_layer_sizes=(50,), max_iter=500)
```

2. Evaluate the model with k-fold cross validation:

```
k_fold = 10
accuracies = cross_val_score(estimator=mlp_classifier, X=X_train, y=y_train, cv=k_fold)

print(f"Accuracy: {accuracies.mean()*100:.2f}% ({accuracies.std()*100:.2f}%)")
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_split.py:700: UserWarn
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/neural_network/_multilayer_perceptron.p
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/neural_network/_multilayer_perceptron.p
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/neural_network/_multilayer_perceptron.p
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/neural_network/_multilayer_perceptron.p
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/neural_network/_multilayer_perceptron.p
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/neural_network/_multilayer_perceptron.p
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/neural_network/_multilayer_perceptron.p
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/neural_network/_multilayer_perceptron.p
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/neural_network/_multilayer_perceptron.p
  warnings.warn(
Accuracy: 7.67% (2.28%)
/usr/local/lib/python3.10/dist-packages/sklearn/neural_network/_multilayer_perceptron.p
  warnings.warn(
```

3. Improve the accuracy of your model by applying additional hidden layers:

```
mlp_classifier = MLPClassifier(hidden_layer_sizes=(50, 50, 50), activation='relu', alpha=0.
mlp_classifier.fit(X_train, y_train)

# Evaluate the model on the test set
y_pred = mlp_classifier.predict(X_test)

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
print("\nAccuracy Score:")
print(accuracy_score(y_test, y_pred))
```

```
          69       0.00      0.00      0.00        10
          70       0.11      0.09      0.10        11
          71       0.00      0.00      0.00         5
          72       0.50      0.20      0.29         5
          73       0.00      0.00      0.00         2
          74       0.00      0.00      0.00         5
          75       0.29      0.28      0.29        18
          76       0.00      0.00      0.00        11
          77       0.33      0.33      0.33         3
          78       0.00      0.00      0.00         5
          79       0.00      0.00      0.00        11
          80       0.29      0.44      0.35         9
          81       0.17      0.12      0.14         8
          82       0.00      0.00      0.00         6
          83       0.00      0.00      0.00         4
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: Unde
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: Unde
  _warn_prf(average, modifier, msg_start, len(result))
```

For regression, do the following:

Create a base model

Improve the model by standardizing the dataset

Show tuning of layers and neurons (see evaluating small and larger networks)

1. Creating a base model

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
```

```
df = pd.read_csv('/content/drive/MyDrive/CPE 019 Emerging Technologies 3/Assignment 7.1/lap
```

```
# Remove unnecessary columns
df = df.drop(['rating', 'Number of Ratings', 'Number of Reviews'], axis=1)

# Convert categorical variables to numerical using one-hot encoding
df = pd.get_dummies(df)

# Split the dataset into features and target variable
X = df.drop('Price', axis=1)
y = df['Price']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
# Define the base model
model = Sequential()
model.add(Dense(64, activation='relu', input_dim=X_train.shape[1]))
model.add(Dense(32, activation='relu'))
model.add(Dense(1, activation='linear'))

# Compile the model
model.compile(loss='mean_squared_error', optimizer='adam')

# Train the model
model.fit(X_train, y_train, epochs=50, batch_size=32, verbose=1)
```

```
21/21 [==============================] - 0s 2ms/step - loss: 4491283456.0000
Epoch 31/50
21/21 [==============================] - 0s 3ms/step - loss: 4259766272.0000
Epoch 32/50
21/21 [==============================] - 0s 3ms/step - loss: 4035402496.0000
Epoch 33/50
21/21 [==============================] - 0s 3ms/step - loss: 3819402240.0000
Epoch 34/50
21/21 [==============================] - 0s 3ms/step - loss: 3611685888.0000
Epoch 35/50
21/21 [==============================] - 0s 3ms/step - loss: 3417762560.0000
Epoch 36/50
21/21 [==============================] - 0s 3ms/step - loss: 3236983552.0000
Epoch 37/50
21/21 [==============================] - 0s 3ms/step - loss: 3066659328.0000
Epoch 38/50
21/21 [==============================] - 0s 3ms/step - loss: 2912227328.0000
Epoch 39/50
21/21 [==============================] - 0s 3ms/step - loss: 2776074240.0000
Epoch 40/50
21/21 [==============================] - 0s 2ms/step - loss: 2651009792.0000
Epoch 41/50
21/21 [==============================] - 0s 4ms/step - loss: 2543217664.0000
Epoch 42/50
21/21 [==============================] - 0s 3ms/step - loss: 2445088512.0000
Epoch 43/50
21/21 [==============================] - 0s 3ms/step - loss: 2360356352.0000
Epoch 44/50
21/21 [==============================] - 0s 3ms/step - loss: 2291592192.0000
Epoch 45/50
21/21 [==============================] - 0s 3ms/step - loss: 2228399616.0000
Epoch 46/50
21/21 [==============================] - 0s 3ms/step - loss: 2174241792.0000
Epoch 47/50
21/21 [==============================] - 0s 4ms/step - loss: 2130108672.0000
Epoch 48/50
21/21 [==============================] - 0s 4ms/step - loss: 2090958208.0000
Epoch 49/50
21/21 [==============================] - 0s 5ms/step - loss: 2056573824.0000
Epoch 50/50
21/21 [==============================] - 0s 4ms/step - loss: 2025516800.0000
<keras.src.callbacks.History at 0x7e9c1e17e9e0>
```

```python
# Standardize the dataset
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Create a new model
model_scaled = Sequential()
model_scaled.add(Dense(64, activation='relu', input_dim=X_train_scaled.shape[1]))
model_scaled.add(Dense(32, activation='relu'))
model_scaled.add(Dense(1, activation='linear'))

# Compile and train the model
model_scaled.compile(loss='mean_squared_error', optimizer='adam')
model_scaled.fit(X_train_scaled, y_train, epochs=50, batch_size=32, verbose=1)
```

```
21/21 [==============================] - 0s 2ms/step - loss: 6667321344.0000
Epoch 30/50
21/21 [==============================] - 0s 2ms/step - loss: 6540859904.0000
Epoch 31/50
21/21 [==============================] - 0s 2ms/step - loss: 6410493952.0000
Epoch 32/50
21/21 [==============================] - 0s 2ms/step - loss: 6277344256.0000
Epoch 33/50
21/21 [==============================] - 0s 2ms/step - loss: 6130328064.0000
Epoch 34/50
21/21 [==============================] - 0s 2ms/step - loss: 5986698752.0000
Epoch 35/50
21/21 [==============================] - 0s 2ms/step - loss: 5835498496.0000
Epoch 36/50
21/21 [==============================] - 0s 2ms/step - loss: 5683389440.0000
Epoch 37/50
21/21 [==============================] - 0s 2ms/step - loss: 5524302336.0000
Epoch 38/50
21/21 [==============================] - 0s 2ms/step - loss: 5364847104.0000
Epoch 39/50
21/21 [==============================] - 0s 2ms/step - loss: 5202641920.0000
Epoch 40/50
21/21 [==============================] - 0s 2ms/step - loss: 5037799424.0000
Epoch 41/50
21/21 [==============================] - 0s 2ms/step - loss: 4871534592.0000
Epoch 42/50
21/21 [==============================] - 0s 2ms/step - loss: 4706433536.0000
Epoch 43/50
21/21 [==============================] - 0s 2ms/step - loss: 4538463232.0000
Epoch 44/50
21/21 [==============================] - 0s 2ms/step - loss: 4374000128.0000
Epoch 45/50
21/21 [==============================] - 0s 2ms/step - loss: 4210573568.0000
Epoch 46/50
21/21 [==============================] - 0s 2ms/step - loss: 4049666304.0000
Epoch 47/50
21/21 [==============================] - 0s 2ms/step - loss: 3890202624.0000
Epoch 48/50
21/21 [==============================] - 0s 2ms/step - loss: 3735452672.0000
Epoch 49/50
21/21 [==============================] - 0s 2ms/step - loss: 3584151040.0000
Epoch 50/50
21/21 [==============================] - 0s 2ms/step - loss: 3439695360.0000
<keras.src.callbacks.History at 0x7e9c0e4560e0>
```

```python
# Define a function to create a model with variable number of layers and neurons
def create_model(layers, neurons):
    model = Sequential()
    model.add(Dense(neurons[0], activation='relu', input_dim=X_train_scaled.shape[1]))

    for i in range(1, layers):
        model.add(Dense(neurons[i], activation='relu'))

    model.add(Dense(1, activation='linear'))

    model.compile(loss='mean_squared_error', optimizer='adam')
    return model

# Define the parameters for tuning
layers = [1, 2, 3]  # Number of hidden layers
neurons = [32, 64, 128]  # Number of neurons in each layer

# Perform grid search to find the best model
best_model = None
best_loss = float('inf')

for layer in layers:
    for neuron in neurons:
        model_tuned = create_model(layer, [neuron] * layer)
        model_tuned.fit(X_train_scaled, y_train, epochs=50, batch_size=32, verbose=0)
        loss = model_tuned.evaluate(X_test_scaled, y_test)

        if loss < best_loss:
            best_loss = loss
            best_model = model_tuned

# Print the best model's architecture and loss
print("Best model architecture:")
best_model.summary()
print("Best model loss:", best_loss)
```

```
6/6 [==============================] - 0s 3ms/step - loss: 7910977536.0000
6/6 [==============================] - 0s 2ms/step - loss: 7841633280.0000
6/6 [==============================] - 0s 3ms/step - loss: 7691759104.0000
6/6 [==============================] - 0s 3ms/step - loss: 4316863488.0000
6/6 [==============================] - 0s 2ms/step - loss: 1712784384.0000
6/6 [==============================] - 0s 3ms/step - loss: 1026023744.0000
6/6 [==============================] - 0s 6ms/step - loss: 820184064.0000
6/6 [==============================] - 0s 2ms/step - loss: 717905728.0000
6/6 [==============================] - 0s 3ms/step - loss: 695052544.0000
Best model architecture:
Model: "sequential_10"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_29 (Dense)            (None, 128)               9344

 dense_30 (Dense)            (None, 128)               16512

 dense_31 (Dense)            (None, 128)               16512

 dense_32 (Dense)            (None, 1)                 129

=================================================================
Total params: 42497 (166.00 KB)
```