ASM1 (P2)

I. Declaring the Library

1.Goal list 2.Goal list

```
import numpy as np
import pandas as pd
from sklearn.datasets import load_diabetes
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import tensorflow as tf
import matplotlib.pyplot as plt
```

```
# Load Diabetes dataset
data = load_diabetes()
```

II. Prepare data

```
# Display properties of data
print("Keys of data:", data.keys())
```

```
Keys of data: dict_keys(['data', 'target', 'frame', 'DESCR', 'feature_names', 'data_filename', 'target_filename', 'data_module'])
```

```
# Show description of data
print("\nDescription of the dataset:")
print(data.DESCR)
```

```
Description of the dataset:
.. _diabetes_dataset:

Diabetes dataset
----------------

Ten baseline variables, age, sex, body mass index, average blood
pressure, and six blood serum measurements were obtained for each of n =
442 diabetes patients, as well as the response of interest, a
quantitative measure of disease progression one year after baseline.

**Data Set Characteristics:**

  :Number of Instances: 442

  :Number of Attributes: First 10 columns are numeric predictive values

  :Target: Column 11 is a quantitative measure of disease progression one year after baseline

  :Attribute Information:
      - age     age in years
      - sex
      - bmi     body mass index
      - bp      average blood pressure
      - s1      tc, total serum cholesterol
      - s2      ldl, low-density lipoproteins
      - s3      hdl, high-density lipoproteins
      - s4      tch, total cholesterol / HDL
      - s5      ltg, possibly log of serum triglycerides level
      - s6      glu, blood sugar level

Note: Each of these 10 feature variables have been mean centered and scaled by the standard deviation times the square root of `n_sa

Source URL:
https://www4.stat.ncsu.edu/~boos/var.select/diabetes.html

For more information see:
Bradley Efron, Trevor Hastie, Iain Johnstone and Robert Tibshirani (2004) "Least Angle Regression," Annals of Statistics (with discu
(https://web.stanford.edu/~hastie/Papers/LARS/LeastAngle_2002.pdf)
```

```
# Displays data of the first few samples
print("\nFirst few samples of the data:")
print(data.data[:5])
```

```
First few samples of the data:
[[ 0.03807591  0.05068012  0.06169621  0.02187239 -0.0442235  -0.03482076
```

```
 -0.04340085 -0.00259226  0.01990749 -0.01764613]
[-0.00188202 -0.04464164 -0.05147406 -0.02632753 -0.00844872 -0.01916334
  0.07441156 -0.03949338 -0.06833155 -0.09220405]
[ 0.08529891  0.05068012  0.04445121 -0.00567042 -0.04559945 -0.03419447
 -0.03235593 -0.00259226  0.00286131 -0.02593034]
[-0.08906294 -0.04464164 -0.01159501 -0.03665608  0.01219057  0.02499059
 -0.03603757  0.03430886  0.02268774 -0.00936191]
[ 0.00538306 -0.04464164 -0.03638469  0.02187239  0.00393485  0.01559614
  0.00814208 -0.00259226 -0.03198764 -0.04664087]]
```

```
# Show labels of the first few samples
print("\nTargets of the first few samples:")
print(data.target[:5])
```

```
Targets of the first few samples:
[151.  75. 141. 206. 135.]
```

```
# Convert data to DataFrame
df = pd.DataFrame(data.data, columns=data.feature_names)
df_target = pd.DataFrame(data.target)

# Use functions like .info() and .shape
print("Shape of the dataframe:", df.shape)
print("\nInformation about the dataframe:")
print(df.info())
```

```
Shape of the dataframe: (442, 10)

Information about the dataframe:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 442 entries, 0 to 441
Data columns (total 10 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   age     442 non-null    float64
 1   sex     442 non-null    float64
 2   bmi     442 non-null    float64
 3   bp      442 non-null    float64
 4   s1      442 non-null    float64
 5   s2      442 non-null    float64
 6   s3      442 non-null    float64
 7   s4      442 non-null    float64
 8   s5      442 non-null    float64
 9   s6      442 non-null    float64
dtypes: float64(10)
memory usage: 34.7 KB
None
```

```
df.head()
```

	age	sex	bmi	bp	s1	s2	s3	s4	
0	0.038076	0.050680	0.061696	0.021872	-0.044223	-0.034821	-0.043401	-0.002592	(
1	-0.001882	-0.044642	-0.051474	-0.026328	-0.008449	-0.019163	0.074412	-0.039493	-(
2	0.085299	0.050680	0.044451	-0.005670	-0.045599	-0.034194	-0.032356	-0.002592	(
3	-0.089063	-0.044642	-0.011595	-0.036656	0.012191	0.024991	-0.036038	0.034309	(
4	0.005383	-0.044642	-0.036385	0.021872	0.003935	0.015596	0.008142	-0.002592	-(

Next steps: View recommended plots

```
df_target.head()
```

	0
0	151.0
1	75.0
2	141.0
3	206.0
4	135.0

```
# Extract features and target
X = data.data
y = data.target

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Standardize features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)


# Build the DNN model

model = tf.keras.Sequential([
    tf.keras.layers.Dense(64, activation='relu', input_shape=(X_train.shape[1],)),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Dense(32, activation='relu'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Dense(1)  # Output layer (1 neuron for regression)
])

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

# Define early stopping callback
early_stopping = tf.keras.callbacks.EarlyStopping(patience=10, restore_best_weights=True)

# Train the model with early stopping
history = model.fit(X_train_scaled, y_train, epochs=300, batch_size=32,
                    validation_split=0.1, callbacks=[early_stopping])

# Plot the loss during training
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()

# Evaluate the model on the test set
mse = model.evaluate(X_test_scaled, y_test)  # Mean Squared Error (MSE)
rmse = np.sqrt(mse)  # Root Mean Squared Error (RMSE)
mae = np.mean(np.abs(model.predict(X_test_scaled) - y_test))  # Mean Absolute Error (MAE)
r_squared = 1 - mse / np.var(y_test)  # R-squared (R^2)

# Print out the evaluation metrics
print("Mean Squared Error (MSE):", mse)
print("Root Mean Squared Error (RMSE):", rmse)
print("Mean Absolute Error (MAE):", mae)
print("R-squared (R^2):", r_squared)


# Save the model to an HDF5 file
model.save("diabetes_model.h5")
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103: UserWarning: You are saving your model as an HDF5 file vi
  saving_api.save_model(
```

```
# Load the model from the file
loaded_model = tf.keras.models.load_model("diabetes_model.h5")

# New data
new_data = np.array([[0.03807591, 0.05068012, 0.06169621, 0.02187235, -0.0442235, -0.03482076,
                      -0.04340085, -0.00259226, 0.01990842, -0.01764613]])

# Standardize the new data
X_new_scaled = scaler.transform(new_data)

# Predict the diabetes progression
predictions = loaded_model.predict(X_new_scaled)
print("Predicted diabetes progression:", predictions[0][0])
```

```
1/1 [==============================] - 0s 98ms/step
Predicted diabetes progression: 231.77246
```

```
predictions
```

```
array([[231.77246]], dtype=float32)
```

ASM1 (M2)

I. Declaring the Library

1. Goal list
2. Goal list

```
import numpy as np
import pandas as pd
from sklearn.datasets import load_diabetes
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import tensorflow as tf
import matplotlib.pyplot as plt

# Load Diabetes dataset
data = load_diabetes()
```

II. Prepare data

```
# Display properties of data
print("Keys of data:", data.keys())
```

```
Keys of data: dict_keys(['data', 'target', 'frame', 'DESCR', 'feature_names', 'data_filename', 'target_filename', 'data_module'])
```

```
# Displays the description of the dataset
print("\nDescription of the dataset:")
print(data.DESCR)
```

```
# Show labels of the first few samples
print("\nTargets of the first few samples:")
print(data.target[:5])
```

```
# Convert data to DataFrame
df = pd.DataFrame(data.data, columns=data.feature_names)
df_target = pd.DataFrame(data.target)

# Use functions like .info() and .shape
print("Shape of the dataframe:", df.shape)
print("\nInformation about the dataframe:")
print(df.info())
```

```
df.head()
```

```
df_target.head()
```

```
# Extract features and target
X = data.data
y = data.target

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Standardize features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

```
# Build the DNN model
model = tf.keras.Sequential([
    tf.keras.layers.Dense(64, activation='selu', kernel_initializer='lecun_normal', input_shape=(X_train.shape[1],)),
    tf.keras.layers.AlphaDropout(0.1),
    tf.keras.layers.Dense(32, activation='selu', kernel_initializer='lecun_normal'),
    tf.keras.layers.AlphaDropout(0.1),
    tf.keras.layers.Dense(1)  # Output layer (1 neuron for regression)
])

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

# Define early stopping callback
early_stopping = tf.keras.callbacks.EarlyStopping(patience=10, restore_best_weights=True)

# Train the model with early stopping
history = model.fit(X_train_scaled, y_train, epochs=700, batch_size=32,
                    validation_split=0.1, callbacks=[early_stopping])

# Plot the loss during training
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()

# Evaluate the model on the test set
mse = model.evaluate(X_test_scaled, y_test)  # Mean Squared Error (MSE)
rmse = np.sqrt(mse)  # Root Mean Squared Error (RMSE)
mae = np.mean(np.abs(model.predict(X_test_scaled) - y_test))  # Mean Absolute Error (MAE)
r_squared = 1 - mse / np.var(y_test)  # R-squared (R^2)

# Print out the evaluation metrics
print("Mean Squared Error (MSE):", mse)
print("Root Mean Squared Error (RMSE):", rmse)
print("Mean Absolute Error (MAE):", mae)
print("R-squared (R^2):", r_squared)


# Save the model to an HDF5 file
model.save("diabetes_model_with_selu.h5")


# Load the model from the file
loaded_model = tf.keras.models.load_model("diabetes_model_with_selu.h5")

# New data
new_data = np.array([[0.03807591, 0.05068012, 0.06169621, 0.02187235, -0.0442235, -0.03482076,
                      -0.04340085, -0.00259226, 0.01990842, -0.01764613]])

# Standardize the new data
X_new_scaled = scaler.transform(new_data)

# Predict the diabetes progression
predictions = loaded_model.predict(X_new_scaled)
print("Predicted diabetes progression:", predictions[0][0])
```

```
predictions
```

ASM1 (D1)

I. Declaring the Library

1. Goal list
2. Goal list

```
import numpy as np
import pandas as pd
from sklearn.datasets import load_diabetes
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import tensorflow as tf
import matplotlib.pyplot as plt

# Load Diabetes dataset
data = load_diabetes()
```

II. Prepare data

```
# Extract features and target
X = data.data
y = data.target

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Standardize features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)


# Build the DNN model with improved architecture
model = tf.keras.Sequential([
    tf.keras.layers.Dense(128, activation='relu', kernel_initializer='he_normal', input_shape=(X_train.shape[1],)),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(64, activation='relu', kernel_initializer='he_normal'),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(32, activation='relu', kernel_initializer='he_normal'),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(1)  # Output layer (1 neuron for regression)
])

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

# Define early stopping callback
early_stopping = tf.keras.callbacks.EarlyStopping(patience=10, restore_best_weights=True)

# Train the model with early stopping
history = model.fit(X_train_scaled, y_train, epochs=700, batch_size=32,
                    validation_split=0.1, callbacks=[early_stopping])

# Plot loss during training
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()

# Evaluate the model on the test set
mse = model.evaluate(X_test_scaled, y_test)  # Mean Squared Error (MSE)
rmse = np.sqrt(mse)  # Root Mean Squared Error (RMSE)
mae = np.mean(np.abs(model.predict(X_test_scaled) - y_test))  # Mean Absolute Error (MAE)
r_squared = 1 - mse / np.var(y_test)  # R-squared (R^2)

# Print out the evaluation metrics
print("Mean Squared Error (MSE):", mse)
print("Root Mean Squared Error (RMSE):", rmse)
print("Mean Absolute Error (MAE):", mae)
print("R-squared (R^2):", r_squared)


# Save the model to an HDF5 file
model.save("diabetes_model_improved.h5")
```