IE406 Machine Learning - Assignment 4

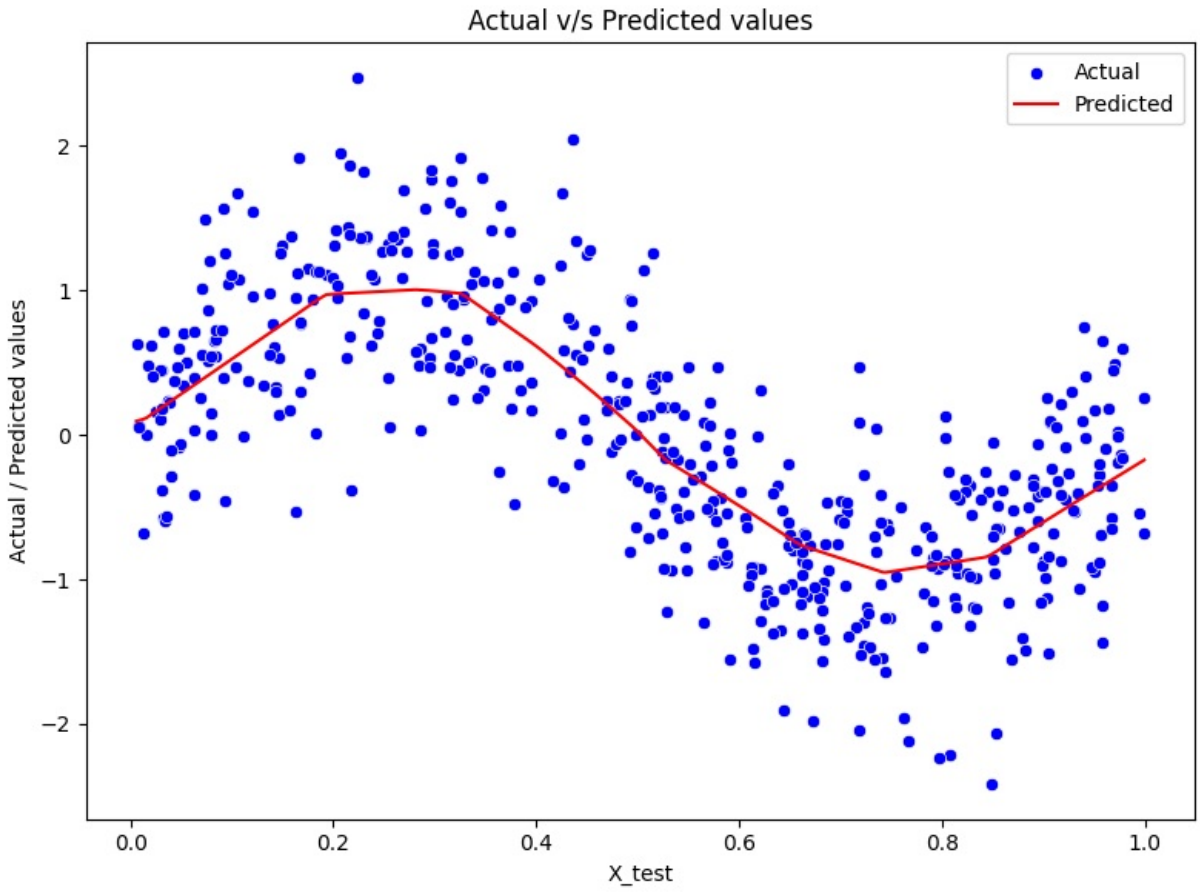
202201140 | Harsh Gajjar

In [6]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from keras import Sequential
from keras.layers import Input, Dense
from keras.models import Model
import seaborn as sns
```

1. Generate 100 real number for the variable X from the uniform distribution U[0,1]. Construct the training set T = {(x1,y1), (x2,y2),……,(x100,y100)} using the relation Yi = sin(2 π xi) + ϵi where ϵi ~ N(0,0.25). In the similar way construct a testing set of size 50 I,e. Test = {(x'1,y'1),(x'2,y'2),……,(x'50,y'50)}. Fit Artificial Neural Network of single hidden layer with 10 number of neurons and plot estimates on testing set. Also compute the RMSE, MAE, MAPE, NMSE and R2.

In [11]:
```
# generating training set (1000 data points for better training)
X_train = np.random.uniform(0, 1, 1000)
epsilon_train = np.random.normal(0, np.sqrt(0.25), 1000)
Y_train = np.sin(2 * np.pi * X_train) + epsilon_train


# generating testing set (500 data points for better testing)
X_test = np.random.uniform(0, 1, 500)
epsilon_test = np.random.normal(0, np.sqrt(0.25), 500)
Y_test = np.sin(2 * np.pi * X_test) + epsilon_test


# Build the ANN Model
input_layer = Input(shape=(1,))
hidden_layer = Dense(128, activation='relu')(input_layer)
hidden_layer = Dense(64, activation='relu')(hidden_layer)
output_layer = Dense(1)(hidden_layer)
model = Model(inputs=input_layer, outputs=output_layer)


# Train and Predict
model.compile(optimizer='adam', loss='mean_squared_error')
model.fit(X_train, Y_train, epochs=200, verbose=0)
Y_pred_test = model.predict(X_test)


# calculating metrics
rmse = np.sqrt(mean_squared_error(Y_test, Y_pred_test))
mae = mean_absolute_error(Y_test, Y_pred_test)
mape = np.mean(np.abs((Y_test - Y_pred_test) / Y_test)) * 100
nmse = mean_squared_error(Y_test, Y_pred_test) / np.var(Y_test)
r2 = r2_score(Y_test, Y_pred_test)

print(f"RMSE: {rmse}")
print(f"MAE: {mae}")
print(f"MAPE: {mape}%")
print(f"NMSE: {nmse}")
print(f"R²: {r2}")


# Plotting
plt.figure(figsize=(8, 6))
sns.scatterplot(x=X_test, y=Y_test, color='blue', label='Actual')
sns.lineplot(x=X_test, y=Y_pred_test.flatten(), color='red', label='Predicted')
plt.title('Actual v/s Predicted values')
plt.xlabel('X_test')
plt.ylabel('Actual / Predicted values')
plt.legend()
plt.tight_layout()
plt.show()
```

```
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
RMSE: 0.5213756125690666
MAE: 0.41371094295237293
MAPE: 429.3275851626344%
NMSE: 0.3422018064898131
R²: 0.6577981935101869
```

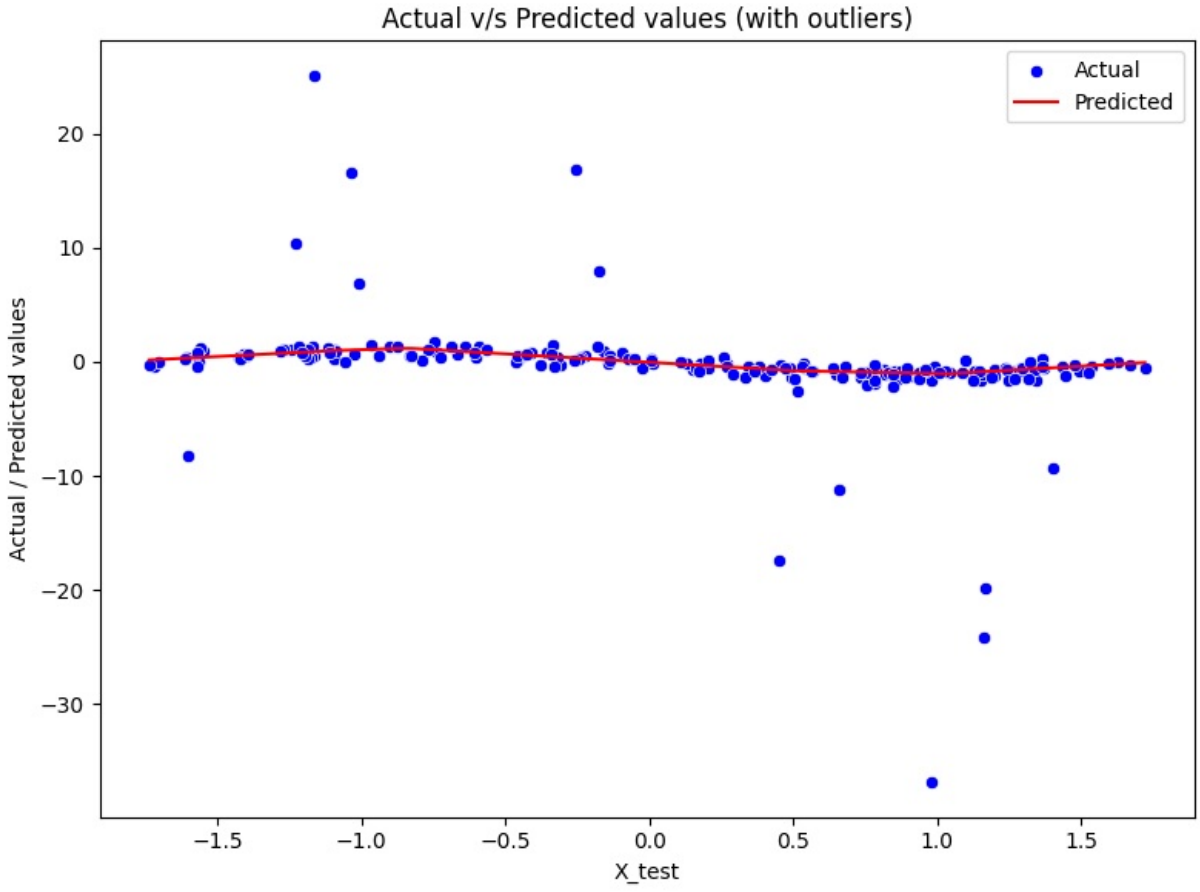


2. Add outliers to training set T : -Modify the training set T by picking up randomly 15 data points from the training set T and scale their yi values by 20. Fit Artificial Neural Network of single hidden layer with 10 number of neurons and plot estimates on testing set. Also compute the RMSE, MAE, MAPE, NMSE and R2. Comment upon the efficiency of least square loss in presence of outliers.

In [12]:
```
# generating training data (1000 data points for better training)
X_train = np.random.uniform(0, 1, 1000).reshape(-1, 1)
epsilon_train = np.random.normal(0, np.sqrt(0.25), 1000)
Y_train = np.sin(2 * np.pi * X_train).flatten() + epsilon_train


# generating testing data (500 data points for better testing)
X_test = np.random.uniform(0, 1, 500).reshape(-1, 1)
epsilon_test = np.random.normal(0, np.sqrt(0.25), 500)
Y_test = np.sin(2 * np.pi * X_test).flatten() + epsilon_test


# Add outliers to the training set
np.random.seed(42)
outlier_indices = np.random.choice(len(Y_train), 15, replace=False)
Y_outliers = Y_train.copy()
Y_outliers[outlier_indices] *= 20


# Split the data into training and testing sets
X_train, X_test, Y_train, Y_test = train_test_split(X_train, Y_outliers, test_size=0.2, random_state=42)


# Standardize the data
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)


# Build the ANN Model
input_layer= Input(shape=(X_train.shape[1],))
hidden_layer = Dense(10, activation='relu')(input_layer)
output_layer = Dense(1)(hidden_layer)
model = Model(inputs=input_layer, outputs=output_layer)


# Train and Predict
model.compile(optimizer='adam', loss='mean_squared_error')
model.fit(X_train, Y_train, epochs=100, batch_size=10, verbose=0)
Y_pred_test = model.predict(X_test)
```

```
# Train and Predict
model.compile(optimizer='adam', loss='mean_squared_error')
model.fit(X_train, Y_train, epochs=200, verbose=0)
Y_pred_test = model.predict(X_test)


# Calculating metrics
rmse = np.sqrt(mean_squared_error(Y_test, Y_pred_test))
mae = mean_absolute_error(Y_test, Y_pred_test)
mape = np.mean(np.abs((Y_test - Y_pred_test) / Y_test)) * 100
nmse = mean_squared_error(Y_test, Y_pred_test) / np.var(Y_test)
r2 = r2_score(Y_test, Y_pred_test)

print(f'RMSE: {rmse}')
print(f'MAE: {mae}')
print(f'MAPE: {mape}%')
print(f'NMSE: {nmse}')
print(f'R²: {r2}')


# Plotting
plt.figure(figsize=(8, 6))
sns.scatterplot(x=X_test[:, 0], y=Y_test, color='blue', label='Actual')
sns.lineplot(x=X_test[:, 0], y=Y_pred_test.flatten(), color='red', label='Predicted')
plt.title('Actual v/s Predicted values (with outliers)')
plt.xlabel('X_test')
plt.ylabel('Actual / Predicted values')
plt.legend()
plt.tight_layout()
plt.show()
```

```
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step
RMSE: 4.494652065705328
MAE: 1.3454152309775658
MAPE: 499.80787586134204%
NMSE: 0.9115271790011886
R²: 0.08847282099881137
```



3. Consider the Boston Housing Dataset. Divide the dataset in training, validation and testing sets. Train ANN using training set, tune parameters of ANN using validation set and evaluate the model on testing set. After tuning the best parameter on validation set, report the RMSE, MAE, MAPE, NMSE and on R2 testing set.

In [14]:
```
# Read the CSV file with space as delimiter
data = pd.read_csv('boston_housing_data2.csv')
X = data.iloc[:, :-1].values
Y = data.iloc[:, -1].values
```

```python
# split the data into training, validation, and testing sets
X_train, X_temp, Y_train, Y_temp = train_test_split(X, Y, test_size=0.3, random_state=42)
X_val, X_test, Y_val, Y_test = train_test_split(X_temp, Y_temp, test_size=0.5, random_state=42)


# Standardize the data
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_val = scaler.transform(X_val)
X_test = scaler.transform(X_test)


# Build the ANN Model
input_layer = Input(shape=(X_train.shape[1],))
hidden_layer = Dense(64, activation='relu')(input_layer)
hidden_layer = Dense(32, activation='relu')(hidden_layer)
output_layer = Dense(1)(hidden_layer)
model = Model(inputs=input_layer, outputs=output_layer)


# Train and Predict
model.compile(optimizer='adam', loss='mse')
model.fit(X_train, Y_train, validation_data=(X_val, Y_val), epochs=200, batch_size=32, verbose=0)
Y_pred_test = model.predict(X_test)


# Calculating errors
rmse = np.sqrt(mean_squared_error(Y_test, Y_pred_test))
mae = mean_absolute_error(Y_test, Y_pred_test)
mape = np.mean(np.abs((Y_test - Y_pred_test) / Y_test)) * 100
nmse = mean_squared_error(Y_test, Y_pred_test) / np.var(Y_test)
r2 = r2_score(Y_test, Y_pred_test)

print(f'RMSE: {rmse}')
print(f'MAE: {mae}')
print(f'MAPE: {mape}%')
print(f'NMSE: {nmse}')
print(f'R2: {r2}')


# Plotting
plt.figure(figsize=(8, 6))
sns.scatterplot(x=X_test[:, 0], y=Y_test, color='blue', label='Actual')
sns.scatterplot(x=X_test[:, 0], y=Y_pred_test.flatten(), color='red', label='Predicted')
plt.title('Actual v/s Predicted values (Boston Housing)')
plt.xlabel('X_test (1st col)')
plt.ylabel('Actual / Predicted values')
plt.legend()
plt.tight_layout()
plt.show()
```

```
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step
RMSE: 4.08845545619004
MAE: 2.3992487267444007
MAPE: 50.0984018534587%
NMSE: 0.21031205348614998
R2: 0.78968794651385
```