David Rosenfeld
UCF ID: 1550340

CAP 5610 – Machine Learning
Spring 2025

Assignment 2

1) Classify the following attributes as nominal, ordinal, interval, or ratio. Explain why.
   a) Rating of an Amazon product on a scale of 1 to 5
      This attribute is **ordinal**. The values have a meaningful order, but the difference between consecutive values (e.g., why a product might be rated as a 4 instead of a 3) is not always clear. In addition, there is no zero value, and mathematical operations are not meaningful.
   b) Internet Speed
      This attribute is a **ratio**. Internet speed is measured in Mbps, which is inherently a ratio unit. It also has a zero value, and mathematical operations are meaningful.
   c) The number of customers in a store
      This attribute is a **ratio**. It is a countable, non-negative integer value, with a zero value. In addition, mathematical operations, including ratio operations, are valid on this value.
   d) UCF Student ID
      This attribute is a **nominal**. It is a unique identifier where ordering and numerical meaning are unimportant. There are no meaningful arithmetic operations and no meaningful comparative operations except equality.
   e) Letter grade (A, B, C, D)
      This attribute is **interval**, but only because the letter grades have specific percentile ranges assigned to them in the course syllabus. When tied to specific percentile ranges, the differences between letter grades are numerically meaningful. There is no true zero (i.e., 0% is not the absence of a grade), and ratios are not meaningful in the sense that 80% is not "twice as much grade" as 40%. Note that if letter grades were not tied to specific percentile ranges, then they would be similar to the Amazon star rating example, and in that case would be **ordinal**.

2) Given the four boxes, answer the following questions:
   a) Which proximity measure would you use to group the boxes based on their shapes (length-width ratio)?
      Given that we are comparing length-width ratios, we are concerned with proportionality rather than absolute size. In this case, **correlation similarity** is an appropriate measure. This measure will give a high correlation for boxes that have a similar proportion in length and width.
   b) Which proximity measure would you use to group the boxes based on their size?
      To group by size, I would use **Euclidean distance**, which compares the difference in size between any two boxes. Compute the Euclidean distance for each of the six possible pairs of boxes (order is unimportant), and then group them by smallest distance.

3) Write Python code to calculate cosine similarity and Euclidean distance using numpy.
   The code for this question is in the appendices at the end of this document. Here is the output:

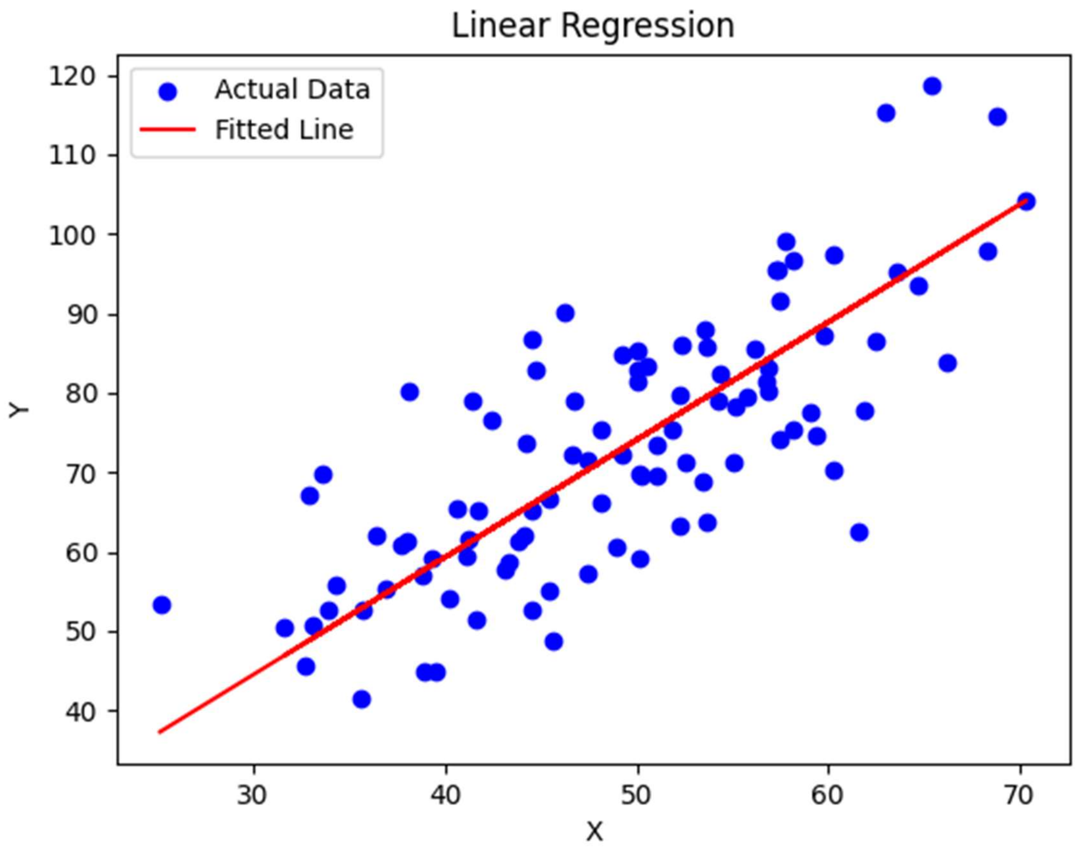
```
Cosine Similarity: 0.9746318461970762
Euclidean Distance: 5.196152422706632
```

4) Implement linear regression to find the best linear model for the provided data. Plot the result using matplotlib.pyplot.
The code for this question is in the appendices at the end of this document. The console output produced during the run is:

```
Epoch 0: MSE = 5611.166153829125
Epoch 100: MSE = 111.05814971560669
Epoch 200: MSE = 111.05287287625939
Epoch 300: MSE = 111.04760379813368
Epoch 400: MSE = 111.04234246981433
Epoch 500: MSE = 111.03708887990277
Epoch 600: MSE = 111.03184301701742
Epoch 700: MSE = 111.02660486979315
Epoch 800: MSE = 111.0213744268818
Epoch 900: MSE = 111.0161516769518

Final slope (m): 1.4796491688881985
Final intercept (c): 0.10148121497503654
```

Here is the plot of the data and the regression line:

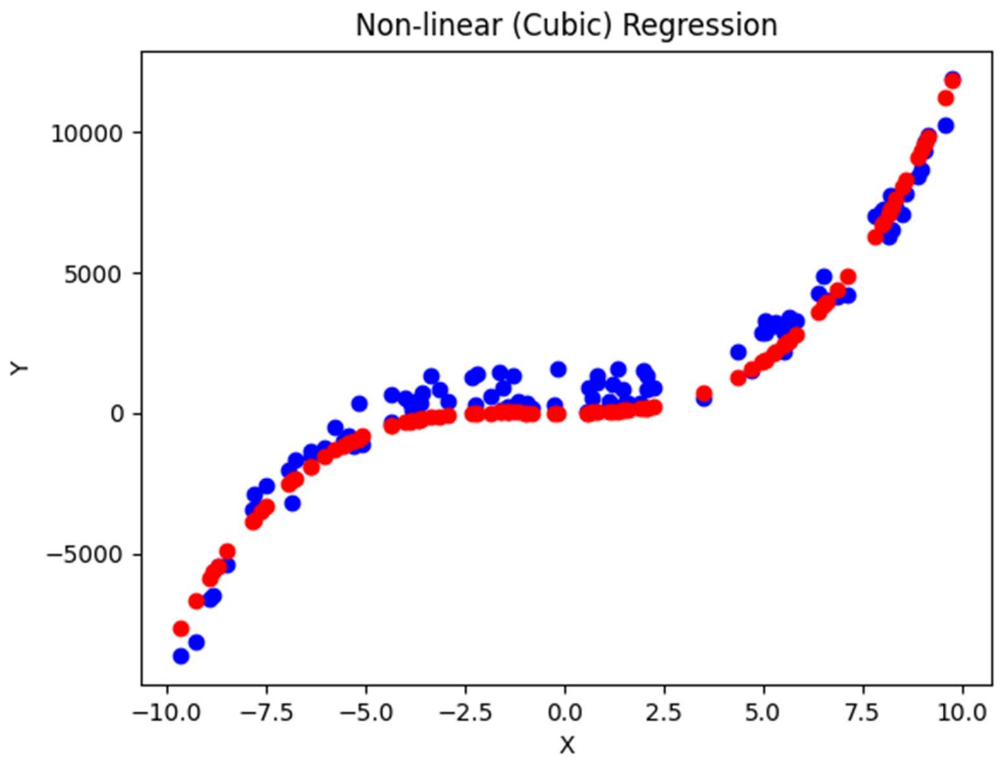


5) Implement non-linear regression to find the best cubic function model for the provided HW2_nonlinear_data.csv. Plot the result.

The code for this question is in the appendices at the end of this document. The console output produced during the run is:

```
Epoch 0: MSE = 17080373.135286063
Epoch 1000: MSE = 593493.028363112
Epoch 2000: MSE = 592151.1002388574
Epoch 3000: MSE = 591529.5514476604
Epoch 4000: MSE = 590910.5835438783
Epoch 5000: MSE = 590293.4908234598
Epoch 6000: MSE = 589678.2549125683
Epoch 7000: MSE = 589064.8583982155
Epoch 8000: MSE = 588453.2841650217
Epoch 9000: MSE = 587843.5153890359
Final coefficients:
  a=10.655168698725843
  b=20.970276527941813
  c=-1.8103041714197277
  d=7.989333607627859
```

Here is the plot of the data and the predicted values after training:



David Rosenfeld
UCF ID: 1550340

Appendices

The following pages contain the source code for questions 3, 4, and 5.

question3.py

```python
import numpy as np

# Calculate cosine similarity.
# Cosine similarity is defined as the dot product of the two vectors,
# divided by the product of the magnitudes of the vectors.
def cosine_similarity(A, B):
    return np.dot(A, B) / (np.linalg.norm(A) * np.linalg.norm(B))

# Calculate Euclidean distance.
# Euclidean distance is the straight-line distance between two points
# in n-dimensional space.
def euclidean_distance(A, B):
    return np.linalg.norm(A - B)

# Declare two fixed, three-dimensional vectors.
v1 = np.array([1, 2, 3])
v2 = np.array([4, 5, 6])

# Compute cosine similarity
cs = cosine_similarity(v1, v2)
# Compute Euclidean distance
ed = euclidean_distance(v1, v2)

# Print results
print("Cosine Similarity:", cs)
print("Euclidean Distance:", ed)

```

question4.py

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# Load data from the provided CSV file.
data = pd.read_csv("HW2_linear_data.csv")
# Load the first (input/feature) column into the vector X.
X = data.iloc[:, 0].values
# Load the second (output/target) colulmn into the vector Y.
Y = data.iloc[:, 1].values

# Initialize the parameters to 0.
# The m value is the slope (i.e., weight), and the c value
# is the y-intercept (i.e., bias).
m = 0
c = 0

# Set learning rate and number of epochs as described in the problem statement.
learning_rate = 0.0001
epochs = 1000

# Count the number of input values.
n = len(X)

# Calculate the gradient descent.
for e in range(epochs):
    # Calculate the predicted value
    Y_pred = m * X + c
    # Calculate the error relative to ground truth.
    error = Y_pred - Y
    # Use the error value to calculate the MSE.
    mse = np.mean(error**2)

    # Update the slope/weight and intercept/bias based on the
    # learning rate and the error.

    # When updating m, the term term "np.sum(error * X)" represents
    # the gradient of the MSE with respect to m.
    m -= learning_rate * (2/n) * np.sum(error * X)  # Update slope
    # When updating c, the term "np.sum(error)" represents the
    # gradient of the MSE with resepct to c.
    c -= learning_rate * (2/n) * np.sum(error)  # Update intercept

    # Print the MSE value every 100 epochs to see changes.
    # The MSE at epoch 0 should be very large because of the initial
    # values of 0 for both m and c, with a sharp drop at epoch
    # epoch 100, followed by small incremental improvements for the
    # remaining epochs.
    if e % 100 == 0:
        print(f"Epoch {e}: MSE = {mse}")

# Generate the final predictions using the trained values of m and c.
Y_pred = m * X + c

# Plot the results.
# Generate a scatter plot of the ground-truth data.
plt.scatter(X, Y, color="blue", label="Actual Data")
# Plot the predictions as a line.
plt.plot(X, Y_pred, color="red", label="Fitted Line")
plt.xlabel("X")
plt.ylabel("Y")
plt.title("Linear Regression")
plt.show()

# Print the final trained parameters.
print(f"Final slope (m): {m}")
print(f"Final intercept (c): {c}")

```

question5.py

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# Load the data from the provided CSV file.
data = pd.read_csv("HW2_nonlinear_data.csv")
# Load the first (input/feature) column into the vector X.
X = data.iloc[:, 0].values  # First column as input (feature)
# Load the second (output/target) colulmn into the vector Y.
Y = data.iloc[:, 1].values  # Second column as output (target)

# Reshape X for computation.  This transforms X from the initial
# one-dimensional array of n values into a 2D array having n rows
# and 1 column.  The first parameter value of -1 causes numpy to
# infer the number of rows based on the size of X.  The second
# parameter gives the number of columns as 1.
X = X.reshape(-1, 1)

# A cubic regression uses an expression of the form:
#     Y = aX^3 + bX^2 + cX + d
# The coefficient values a, b, c, and d are the values that will
# be learned through the training.

# Initialize the coefficients to zero.
a = b = c = d = 0

# Set the learning rate and number of epochs to the values given in
# the problem statement.  They can be adjusted, but start here.
learning_rate = 1e-6
epochs = 10000

# Count the number of input values.
n = len(X)

# Calculate the gradient descent.
for e in range(epochs):
    # Calcuate the predicted value
    Y_pred = (a*(X**3)) + (b*(X**2)) + (c*X) + d
    # Calculate the error relative to ground truth.
    error = Y_pred.flatten() - Y
    # Use the error value to calculate the MSE.
    mse = np.mean(error**2)

    # Update the coefficients using gradient descent.

    # Compute gradients, which are the partial derivatives
    # with respect to a, b, c, d).
    # The flatten() method takes the 2D array created by reshape()
    # and returns 1D array that is needed for numpy element-wise operations.
    da = (2/n) * np.sum(error * X.flatten()**3)
    db = (2/n) * np.sum(error * X.flatten()**2)
    dc = (2/n) * np.sum(error * X.flatten())
    dd = (2/n) * np.sum(error)

    # Update the coefficients using the learning rate and the gradients.
    a -= learning_rate * da
    b -= learning_rate * db
    c -= learning_rate * dc
    d -= learning_rate * dd

    # Print the MSE value every 1000 epochs to see changes.
    if e % 1000 == 0:
        print(f"Epoch {e}: MSE = {mse}")

# Generate the final predictions using the trained coefficients.
Y_pred = (a*(X**3)) + (b*(X**2)) + (c*X) + d

# Plot the results
# Generate a scatter plot of the ground-truth data.
plt.scatter(X, Y, color="blue", label="Actual Data")
# Overlay the predicted values as red points.
plt.scatter(X, Y_pred, color="red", label="Fitted Curve")
plt.xlabel("X")
plt.ylabel("Y")
plt.title("Non-linear (Cubic) Regression")
plt.show()

# Print final trained coefficients.
print(f"Final coefficients:\n     a={a}\n     b={b}\n     c={c}\n     d={d}")

```