CAP_5610_Assignment_2_Solution_Arman_Sayan

February 4, 2025

CAP 5610 Assignment #2: Data and Linear Regression

This source code is written by Arman Sayan.

Last Edit: February 4, 2024

1 Q3 - Coding Question

Please write a Python code to calculate Cosine similarity, and Euclidean distance using NumPy. The input can be two randomly generated vectors or fixed vectors written by yourself.

1.1 Answer:

The formula for the cosine similarity is

$$CS = \frac{A \cdot B}{||A|| \cdot ||B||}$$

where $A \cdot B$ is the dot product of vectors A and B, $||A||$ is the magnitude of vector A, and $||B||$ is the magnitude of vector B.

[1]:
```
# Import necessary libraries
import numpy as np
import random as rnd
import math
```

[2]:
```
# Define a function to calculate Cosine similarity
# using Numpy
def CosineSimilarity_v1(A, B):
  # Calculate norm/magnitude of each vector
  norm_vecA = np.linalg.norm(A)
  norm_vecB = np.linalg.norm(B)

  if norm_vecA == 0 or norm_vecB == 0:
    return None  # to indicate undefined similarity

  # Calculate dot product
  dot_product = np.dot(A, B)

  # Combine results
```

```
    return dot_product / (norm_vecA * norm_vecB)
```

[3]:
```
# Define a function to calculate Cosine similarity
# without Numpy
def CosineSimilarity_v2(A, B):
  # Calculate norm/magnitude of each vector
  norm_vecA = math.sqrt(sum(a_i ** 2 for a_i in A))
  norm_vecB = math.sqrt(sum(b_i ** 2 for b_i in B))

  if norm_vecA == 0 or norm_vecB == 0:
    return None  # to indicate undefined similarity

  # Calculate dot product
  dot_product = sum(a_i * b_i for a_i, b_i in zip(A, B))

  # Combine results
  return dot_product / (norm_vecA * norm_vecB)
```

The formula for the Eucledian distance is

$$|A - B| = \sqrt{\sum_{i=1}^{n} (a_i - b_i)^2}$$

where n is the length of the vector.

[4]:
```
# Define a function to calculate Euclidean distance
# using Numpy
def EuclideanDistance_v1(A, B):
  return np.linalg.norm(A - B)
```

[5]:
```
# Define a function to calculate Euclidean distance
# without Numpy
def EuclideanDistance_v2(A, B):
  distance = 0
  for i in range(len(A)):
    distance += (A[i] - B[i]) ** 2
  return distance ** 0.5
```

[6]:
```
# Learn the vector size from the user
inputCollected = False
vectorSize = 0
while not inputCollected:
  try:
    vectorSize = int(input("Enter the vector size: "))
    if vectorSize <= 0:
      raise ValueError
    maxValue = int(input("Enter the maximum value of the vector elements: "))
    if maxValue <= 0:
```

```
            raise ValueError
        inputCollected = True
    except ValueError:
        print("Invalid input. Please enter a valid integer.")

# Generate two random vectors based on the vector size
vector1 = np.random.randint(low=-maxValue, high=maxValue, size=vectorSize)
vector2 = np.random.randint(low=-maxValue, high=maxValue, size=vectorSize)

# Print the results
print("Vector 1:", vector1)
print("Vector 2:", vector2)
print("Cosine Similarity with Numpy:", CosineSimilarity_v1(vector1, vector2))
print("Cosine Similarity without Numpy:", CosineSimilarity_v2(vector1, vector2))
print("Euclidean Distance with Numpy:", EuclideanDistance_v1(vector1, vector2))
print("Euclidean Distance without Numpy:", EuclideanDistance_v2(vector1,␣
↪vector2))
```

```
Enter the vector size: 5
Enter the maximum value of the vector elements: 2
Vector 1: [ 0 -2 -1  1  0]
Vector 2: [ 0  0  0 -1  0]
Cosine Similarity with Numpy: -0.4082482904638631
Cosine Similarity without Numpy: -0.4082482904638631
Euclidean Distance with Numpy: 3.0
Euclidean Distance without Numpy: 3.0
```

2 Q4 - Coding Question

Please implement a Linear Regression to find the best linear model for the provided linear data. Please plot the result using ”matplotlib.pyplot”.

2.1 Answer:

[7]:
```
# Import necessary libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

[8]:
```
# Load HW2_linear_data.csv and preprocess data
data = pd.read_csv("HW2_linear_data.csv")
X = data.iloc[:, 0].values  # treat the first column as the input feature
y = data.iloc[:, 1].values  # treat the second column as the target variable

# Reshape X and y for computations
X = X.reshape(-1, 1)
y = y.reshape(-1, 1)
```

[9]:
```python
# Initialize necessary parameters that will be used
m = 0  # Slope coefficient
c = 0  # Intercept value
learning_rate = 0.0001  # Fixed learning rate
epochs = 1000  # Number of iterations
N = len(y)  # Number of data points
```

For a dataset with N samples, the MSE is:

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - (\hat{m}x_i + \hat{c}))^2$$

where y_i is the actual target value, $\hat{m}x_i + \hat{c}$ is the predicted value.

$\hat{m}$ the slope, and $\hat{c}$ the intercept are the parameters we want to optimize.

Then, we need to take the derivative of MSE with respect to each of these parameters:

$$\frac{\partial MSE}{\partial m} = -\frac{2}{N}\sum_{i=1}^{N}x_i(y_i - (\hat{m}x_i + \hat{c}))$$

$$\frac{\partial MSE}{\partial c} = -\frac{2}{N}\sum_{i=1}^{N}(y_i - (\hat{m}x_i + \hat{c}))$$

[10]:
```python
# Perform Gradient Descent algorithm
for epoch in range(epochs):
  y_pred = m * X + c  # Compute predictions
  error = y_pred - y  # Compute error

  # Compute gradients
  DwrtM = (2 / N) * np.sum(error * X)  # Derivative wrt m
  DwrtC = (2 / N) * np.sum(error)  # Derivative wrt c

  # Update parameters
  m -= learning_rate * DwrtM
  c -= learning_rate * DwrtC

  # Print loss in every 50 iterations
  if epoch % 50 == 0:
    mse = np.mean(error ** 2)
    print(f"For epoch {epoch}, MSE: {mse:.6f}")
```

```
For epoch 0, MSE: 5611.166154
For epoch 50, MSE: 111.060791
For epoch 100, MSE: 111.058150
For epoch 150, MSE: 111.055510
For epoch 200, MSE: 111.052873
For epoch 250, MSE: 111.050237
For epoch 300, MSE: 111.047604
For epoch 350, MSE: 111.044972
```

```
For epoch 400, MSE: 111.042342
For epoch 450, MSE: 111.039715
For epoch 500, MSE: 111.037089
For epoch 550, MSE: 111.034465
For epoch 600, MSE: 111.031843
For epoch 650, MSE: 111.029223
For epoch 700, MSE: 111.026605
For epoch 750, MSE: 111.023989
For epoch 800, MSE: 111.021374
For epoch 850, MSE: 111.018762
For epoch 900, MSE: 111.016152
For epoch 950, MSE: 111.013543
```

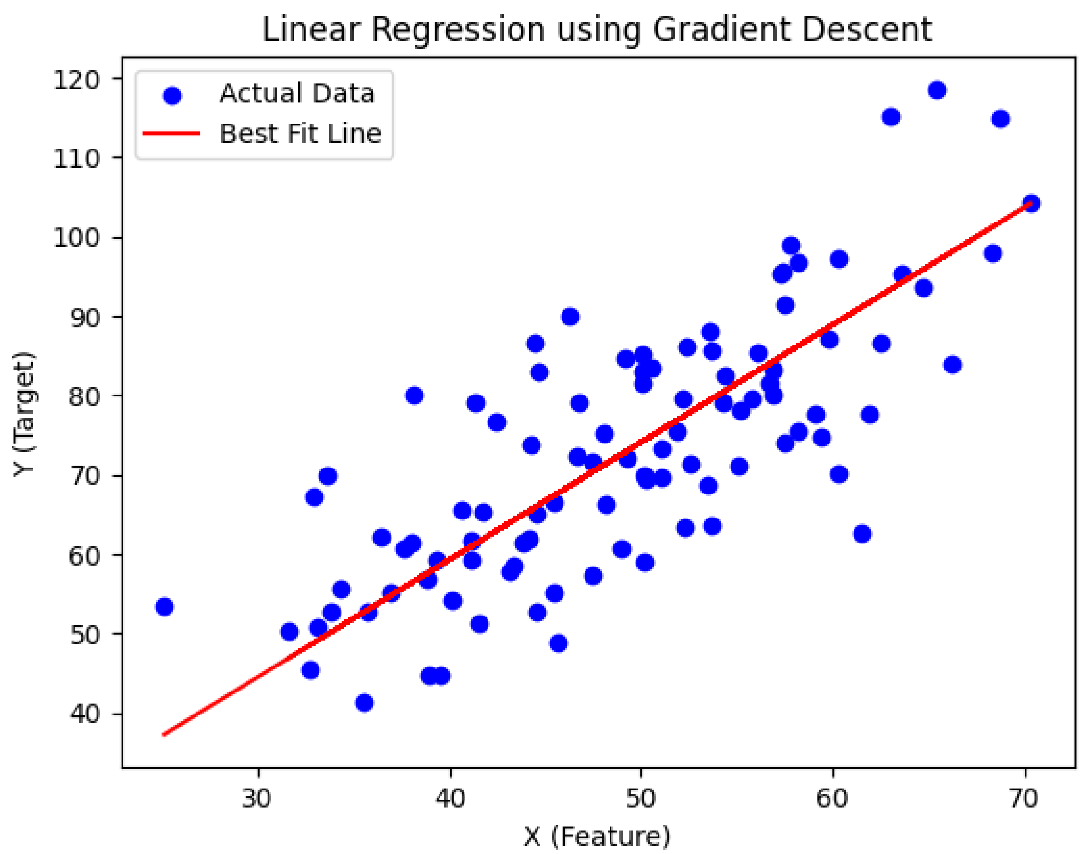
[11]:

```python
# Final linear model
print(f"Final model: Y = {m:.3f}X + {c:.3f}")
```

```
Final model: Y = 1.480X + 0.101
```

[12]:

```python
# Plot the data
plt.scatter(X, y, color="blue", label="Actual Data")
plt.plot(X, m * X + c, color="red", label="Best Fit Line")
plt.xlabel("X (Feature)")
plt.ylabel("Y (Target)")
plt.title("Linear Regression using Gradient Descent")
plt.legend()
plt.show()
```



3 Q5 - Coding Question

Please implement a non-linear regression to find the best cubic function model for the provided non-linear data. Please plot the result, too.

3.1 Answer:

[13]:
```
# Import necessary libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

[14]:
```
# Load HW2_linear_data.csv and preprocess data
data = pd.read_csv("HW2_nonlinear_data.csv")
X = data.iloc[:, 0].values  # treat the first column as the input feature
y = data.iloc[:, 1].values  # treat the second column as the target variable

# Expand X to include cubic terms
X_poly = np.vstack([X**3, X**2, X, np.ones_like(X)]).T
```

[15]:
```python
# Initialize necessary parameters and hyperparameters that will be used
a, b, c, d = np.random.randn(4)  # Random initialization
params = np.array([a, b, c, d])
learning_rate = 0.000001  # Fixed learning rate
epochs = 10000  # Number of iterations
N = len(X)  # Number of data points
```

[16]:
```python
# Perform Gradient Descent algorithm
for epoch in range(epochs):
  y_pred = X_poly @ params  # Compute predictions
  errors = y_pred - y  # Compute error

  # Compute gradients
  gradients = (2 / N) * (X_poly.T @ errors)

  # Update parameters
  params -= learning_rate * gradients

  # Print loss in every 500 iterations
  if epoch % 500 == 0:
    mse = np.mean(errors ** 2)
    print(f"For epoch {epoch}, MSE: {mse:.6f}")
```

```
For epoch 0, MSE: 19202881.150963
For epoch 500, MSE: 619216.402912
For epoch 1000, MSE: 592605.881596
For epoch 1500, MSE: 591483.788087
For epoch 2000, MSE: 591151.065418
For epoch 2500, MSE: 590843.210586
For epoch 3000, MSE: 590536.558239
For epoch 3500, MSE: 590230.373919
For epoch 4000, MSE: 589924.632958
For epoch 4500, MSE: 589619.332675
For epoch 5000, MSE: 589314.471084
For epoch 5500, MSE: 589010.046238
For epoch 6000, MSE: 588706.056208
For epoch 6500, MSE: 588402.499078
For epoch 7000, MSE: 588099.372951
For epoch 7500, MSE: 587796.675946
For epoch 8000, MSE: 587494.406197
For epoch 8500, MSE: 587192.561853
For epoch 9000, MSE: 586891.141080
For epoch 9500, MSE: 586590.142058
```

[17]:
```python
# Extract learned parameters
a, b, c, d = params
print(f"Learned coefficients: a={a:.6f}, b={b:.6f}, c={c:.6f}, d={d:.6f}")
```

```
# Final non-linear model
print(f"Final model: Y = {a:.3f}X^3 + {b:.3f}X^2 + {c:.3f}X + {d:.3f}")
```

```
Learned coefficients: a=10.675649, b=20.947880, c=-3.191184, d=8.925191
Final model: Y = 10.676X^3 + 20.948X^2 + -3.191X + 8.925
```

[18]:

```
# Generate predictions for visualization
X_sorted = np.sort(X)
Y_fitted = a * X_sorted**3 + b * X_sorted**2 + c * X_sorted + d

# Plot results
plt.scatter(X, y, color='blue', label='Original Data')
plt.plot(X_sorted, Y_fitted, color='red', linewidth=2, label='Fitted Cubic␣
↪Model')
plt.xlabel("X (Feature)")
plt.ylabel("Y (Target)")
plt.title("Cubic Regression using Gradient Descent")
plt.legend()
plt.show()
```