

AI VIET NAM – COURSE 2024

Module 4 - Exercise 2

Linear Regression

Ngày 12 tháng 10 năm 2024

Giới thiệu về bài tập: Ở phần bài tập này các bạn sẽ được ôn tập về giải thuật linear regression cho bài toán advertising theo cách vectorization dùng stochastic gradient descent, m samples (mini-batch gradient descent), and N samples (batch gradient descent).

Bài tập 1 (Hiện thực giải thuật linear regression theo phương pháp vectorization):

Linear Regression: Các bạn thực hiện train linear regression model trên tập data advertising.csv theo các yêu cầu sau. Các bạn sẽ dựa trên 3 thông tin đầu vào là TV, Radio, Newspaper để dự đoán Sale.

Giới thiệu về tập data: Data có 200 samples (rows), gồm 4 cột thông tin Tv, Radio, Newspaper, và Sales. Đề bài yêu cầu dùng thông tin ở 3 cột đầu tiên (Tv, Radio, Newspaper) để dự đoán được cột cuối cùng (Sale) dùng linear regression model. Dữ liệu Advertising có thể được tải về [tại đây](#).

Để chuẩn hoá data đầu vào, AIVN cung cấp trước cho các bạn function đọc dữ liệu và chuẩn hoá `mean_normalization(X)` như bên dưới:

```
1 # dataset
2 data = genfromtxt('advertising.csv', delimiter=',', skip_header=1)
3 N = data.shape[0]
4 X = data[:, :3]
5 y = data[:, 3:]
6
7 # Normalize input data by using mean normalizaton
8 def mean_normalization(X):
9     N = len(X)
10    maxi = np.max(X)
11    mini = np.min(X)
12    avg = np.mean(X)
13    X = (X - avg) / (maxi - mini)
14    X_b = np.c_[np.ones((N, 1)), X]
15    return X_b, maxi, mini, avg
16
17 X_b, maxi, mini, avg = mean_normalization(X)
```

Yêu cầu của bài tập này là các bạn lần lượt hiện thực lại giải thuật linear regression để dự đoán Sales dựa vào các yêu cầu sau:

1. Hoàn thành fuction `stochastic_gradient_descent()` để huấn luyện data sử dụng Stochastic Gradient Descent. Lưu ý các bạn cần tận dụng tối đa vectorization để hoàn thiện bài tập này.

- **input:** (4 inputs) `X_b`, `y`, `n_epochs`, `learning_rate`
- **output:** `thetas_path`, `losses`

```

1  def stochastic_gradient_descent(X_b, y, n_epochs=50, learning_rate=0.00001):
2
3  # thetas = np.random.randn(4, 1) # uncomment this line for real application
4  thetas = np.asarray([[1.16270837], [-0.81960489], [1.39501033],
5  [0.29763545]])
6
7  thetas_path = [thetas]
8  losses = []
9
10 for epoch in range(n_epochs):
11     for i in range(N):
12         # select random number in N
13         # random_index = np.random.randint(N) #In real application, you
14         # should use this code
15         random_index = i # This code is used for this assignment only
16
17         xi = X_b[random_index:random_index+1]
18         yi = y[random_index:random_index+1]
19
20         # Compute output
21         *****Your code here *****
22
23         # Compute loss li
24         *****Your code here *****
25
26         # Compute gradient for loss
27         *****Your code here *****
28
29         # Compute gradient
30         *****Your code here *****
31
32         # update theta
33         *****Your code here *****
34
35         # logging
36         *****Your code here *****
37
38 return thetas_path, losses

```

Hình 1 là kết quả sau khi thực thi đoạn code sau:

```

1  sgd_theta, losses = stochastic_gradient_descent(X_b, y, n_epochs=50,
2  learning_rate=0.01 )
3
4  x_axis = list(range(500))
5  plt.plot(x_axis, losses[:500], color="r")
6  plt.show()

```

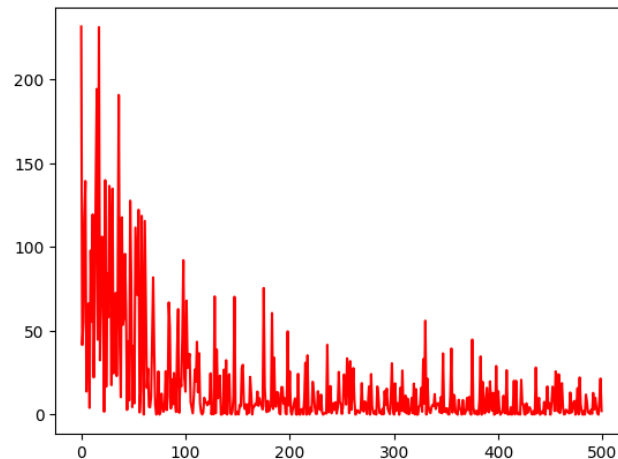
Question 1: Kết quả của đoạn code sau đây:

```

1  sgd_theta, losses = stochastic_gradient_descent(X_b, y, n_epochs=1, learning_rate
2  =0.01 )
3  print(np.sum(losses))

```

- a) 7754.64
- b) 6754.64
- c) 8754.64



Hình 1: Kết quả loss values sử dụng Stochastic Gradient Descent

d) 9754.64

2. Hoàn thành function `mini_batch_gradient_descent()` để huấn luyện data sử dụng Mini-batch Gradient Descent. Lưu ý các bạn cần tận dụng tối đa vectorization để hoàn thiện bài tập này.

- **input:** (5 inputs) `X_b`, `y`, `n_epochs`, `minibatch_size`, `learning_rate`
- **output:** `thetas_path`, `losses`

```

1      def mini_batch_gradient_descent(X_b, y, n_epochs=50, minibatch_size =
2      20, learning_rate=0.01):
3
4      # thetas = np.random.randn(4, 1)
5      thetas = np.asarray([[1.16270837], [-0.81960489], [1.39501033],
6      [0.29763545]])
7
8      thetas_path = [thetas]
9      losses = []
10
11     for epoch in range(n_epochs):
12         # shuffled_indices = np.random.permutation(N) # uncomment this code for
13         # real application
14
15         shuffled_indices = np.asarray([21, 144, 17, 107, 37, 115, 167, 31, 3,
16         132, 179, 155, 36, 191, 182, 170, 27, 35, 162, 25, 28, 73, 172, 152, 102, 16,
17         185, 11, 1, 34, 177, 29, 96, 22, 76, 196, 6, 128, 114, 117, 111, 43, 57, 126,
18         165, 78, 151, 104, 110, 53, 181, 113, 173, 75, 23, 161, 85, 94, 18, 148, 190,
19         169, 149, 79, 138, 20, 108, 137, 93, 192, 198, 153, 4, 45, 164, 26, 8, 131,
20         77, 80, 130, 127, 125, 61, 10, 175, 143, 87, 33, 50, 54, 97, 9, 84, 188, 139,
21         195, 72, 64, 194, 44, 109, 112, 60, 86, 90, 140, 171, 59, 199, 105, 41, 147,
22         92, 52, 124, 71, 197, 163, 98, 189, 103, 51, 39, 180, 74, 145, 118, 38, 47,
23         174, 100, 184, 183, 160, 69, 91, 82, 42, 89, 81, 186, 136, 63, 157, 46, 67,
24         129, 120, 116, 32, 19, 187, 70, 141, 146, 15, 58, 119, 12, 95, 0, 40, 83, 24,
25         168, 150, 178, 49, 159, 7, 193, 48, 30, 14, 121, 5, 142, 65, 176, 101, 55,
26         133, 13, 106, 66, 99, 68, 135, 158, 88, 62, 166, 156, 2, 134, 56, 123, 122,
27         154])

```

```

15     X_b_shuffled = X_b[shuffled_indices]
16     y_shuffled = y[shuffled_indices]
17
18     for i in range(0, N, minibatch_size):
19         xi = X_b_shuffled[i:i+minibatch_size]
20         yi = y_shuffled[i:i+minibatch_size]
21
22         # compute output
23         *****Your code here *****
24
25         # compute loss
26         *****Your code here *****
27
28         # compute derivative of loss
29         *****Your code here *****
30
31         # compute derivative of parameters
32         *****Your code here *****
33
34         # update parameters
35         thetas = thetas - learning_rate*gradients
36         thetas_path.append(thetas)
37
38         loss_mean = np.sum(loss)/minibatch_size
39         losses.append(loss_mean)
40
41     return thetas_path, losses
42

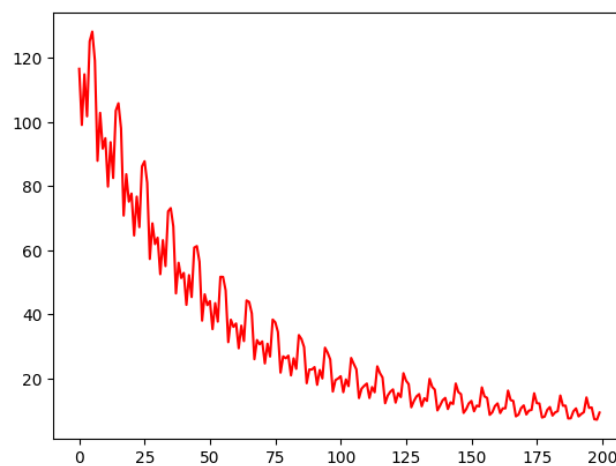
```

Hình 2 là kết quả sau khi thực thi đoạn code sau:

```

1 mbgd_thetas, losses = mini_batch_gradient_descent(X_b, y, n_epochs=50,
2             minibatch_size = 20, learning_rate=0.01)
3 x_axis = list(range(200))
4 plt.plot(x_axis, losses[:200], color="r")
5 plt.show()

```



Hình 2: Kết quả loss values sử dụng Mini batch Gradient Descent

Question 2: Kết quả của đoạn code sau đây:

```

1 mbgd_thetas, losses = mini_batch_gradient_descent(X_b, y, n_epochs=50,
    minibatch_size = 20, learning_rate=0.01)
2 print(round(sum(losses),2))

```

- a) 7865.65
- b) 6865.65
- c) 5865.65
- d) 8865.65

3. Hoàn thành function **batch_gradient_descent()** để huấn luyện data sử dụng batch Gradient Descent. Lưu ý các bạn cần tận dụng tối đa vectorization để hoàn thiện bài tập này.

- **input:** (4 inputs) X_b, y, n_epochs, learning_rate
- **output:** thetas_path, losses

```

1 def batch_gradient_descent(X_b, y, n_epochs=100, learning_rate=0.01):
2
3     # thetas = np.random.randn(4, 1) # uncomment this line for real application
4     thetas = np.asarray([[1.16270837], [-0.81960489], [1.39501033],
5                          [0.29763545]])
6
7     thetas_path = [thetas]
8     losses = []
9
10    for i in range(n_epochs):
11        # compute output
12        *****Your code here *****
13
14        # Compute loss
15        *****Your code here *****
16
17
18        # Compute losses's derivative
19        *****Your code here *****
20
21
22        # computer parameters' derivative
23        *****Your code here *****
24
25        # Update parameters
26        thetas = thetas - learning_rate*gradients
27        thetas_path.append(thetas)
28
29        mean_loss = np.sum(loss)/N
30        losses.append(mean_loss)
31
32    return thetas_path, losses

```

Hình 3 là kết quả sau khi thực thi đoạn code sau:

```

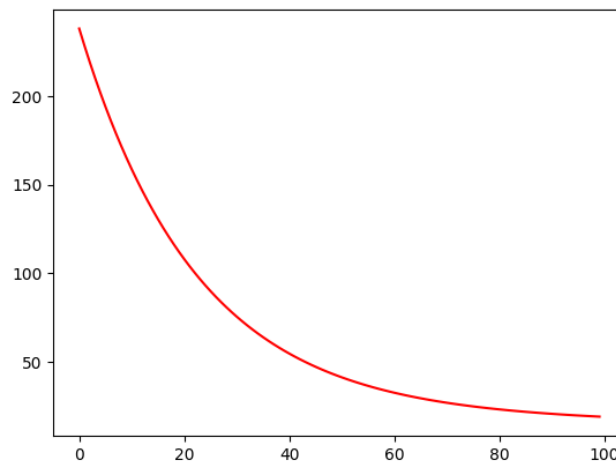
1 bgd_thetas, losses = batch_gradient_descent(X_b, y, n_epochs=100, learning_rate
    =0.01)
2
3 x_axis = list(range(100))

```

```

4 plt.plot(x_axis, losses[:100], color="r")
5 plt.show()

```



Hình 3: Kết quả loss values sử dụng batch Gradient Descent

Question 3: Kết quả của đoạn code sau đây:

```

1 bgd_thetas, losses = batch_gradient_descent(X_b, y, n_epochs=100, learning_rate
    =0.01)
2 print(round(sum(losses),2))

```

- a) 7716.46
- b) 8716.46
- c) 6716.46
- d) 5716.46

NOTE: YÊU CẦU CỦA ĐỀ BÀI LÀ PHẢI THỰC HIỆN THEO VECTORIZATION

- **X_b**: là thông tin Tv, Radio, Newspaper (thông tin model nhận vào và sử dụng để predict Sale) (đã được normalize)
- **y**: là thông tin Sale (thông tin mong muốn model dự đoán đúng)
- **n_epochs**: Số lần train toàn bộ sample trong data
- **minibatch_size**: Số lượng sample sẽ được train trong 1 step (Chỉ sử dụng ở câu b)
- **learning_rate**: Tốc độ học
- **thetas_path**: List weights của model từ lúc khởi tạo cho đến sau mỗi lần cập nhật weights
- **losses**: List loss của mỗi step sau khi cập nhật

Bài tập 2 (Bitcoin forecasting):

1. **Tải bộ dữ liệu:** Các bạn tải bộ dữ liệu BTC-Daily.csv tại [đây](#).
2. **Đọc bộ dữ liệu:** Sử dụng thư viện pandas, chúng ta sẽ đọc file .csv lên như sau:

```

1 # Load dataset
2 import pandas as pd
3
4 df = pd.read_csv('./BTC-Daily.csv')
5
6 # Remove duplicate rows
7 df = df.drop_duplicates()

```

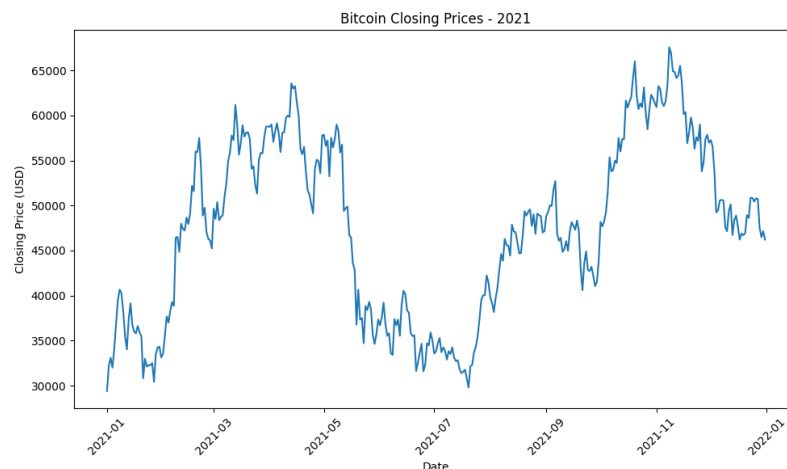
3. **Phân tích dữ liệu:** Thống kê giá kết thúc phiên qua các năm trong bộ dữ liệu

Question 4: Sau khi đoạn code bên dưới được hoàn thiện chính xác, có bao nhiêu biểu đồ thể hiện giá qua các năm ?

```

1 # Range of dates covered
2 df['date'] = pd.to_datetime(df['date'])
3 date_range = str(df['date'].dt.date.min()) + ' to ' + str(df['date'].dt.date.max())
4 print(date_range)
5
6 ##### Your code here #####
7
8 for year in unique_years:
9     ##### Your code here #####
10
11     merged_data = pd.merge(year_month_day, df, on=['year', 'month', 'day'], how='
left')
12     # Plot
13     plt.figure(figsize=(10, 6))
14     plt.plot(merged_data['date_x'], merged_data['close'])
15     plt.title(f'Bitcoin Closing Prices - {year}')
16     plt.xlabel('Date')
17     plt.ylabel('Closing Price (USD)')
18     plt.xticks(rotation=45)
19     plt.tight_layout()
20     plt.show()

```



Hình 4: Biểu đồ giá kết thúc phiên trong năm 2021

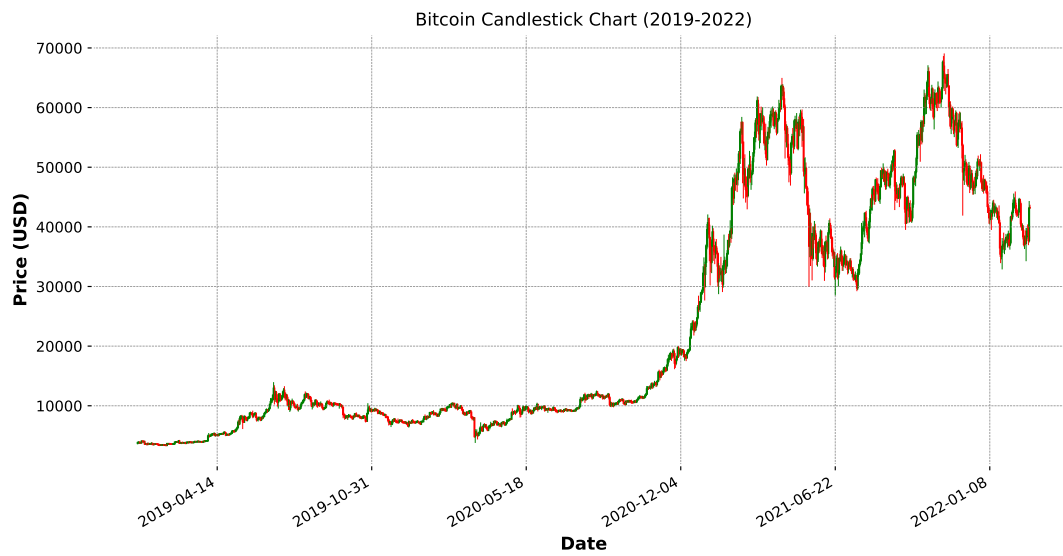
- a) 6
- b) 7
- c) 8
- d) 9

Biểu đồ nến giá giao dịch từ năm 2019 - 2022:

Cài đặt thư viện mplfinance để vẽ biểu đồ nến giao dịch

```
1 !pip install mplfinance

1 import matplotlib.pyplot as plt
2 import matplotlib.dates as mdates
3 from mplfinance.original_flavor import candlestick_ohlc
4 import datetime
5
6 # Filter data for 2019-2022
7 df_filtered = df[(df['date'] >= '2019-01-01') & (df['date'] <= '2022-12-31')]
8
9 # Convert date to matplotlib format
10 df_filtered['date'] = df_filtered['date'].map(mdates.date2num)
11
12 # Create the candlestick chart
13 fig, ax = plt.subplots(figsize=(20, 6))
14
15 candlestick_ohlc(ax, df_filtered[['date', 'open', 'high', 'low', 'close']].values
16     , width=0.6, colorup='g', colordown='r')
17
18 ax.xaxis.set_major_formatter(mdates.DateFormatter('%Y-%m-%d'))
19 fig.autofmt_xdate()
20
21 plt.title('Bitcoin Candlestick Chart (2019-2022)')
22 plt.xlabel('Date')
23 plt.ylabel('Price (USD)')
24 plt.grid(True)
25
26 # Save the plot as a PDF
27 plt.savefig('bitcoin_candlestick_2019_2022.pdf')
28 plt.show()
```

Hình 5: Biểu đồ giao dịch từ năm 2019 - 2022

4. Tạo mô hình Linear Regression:

```

1 def predict(X, w, b):
2     ##### Your code here #####
3
4 def gradient(y_hat, y, x):
5     ##### Your code here #####
6     return (dw, db, cost)
7
8 def update_weight(w, b, lr, dw, db):
9     ##### Your code here #####
10    return (w_new, b_new)

```

Question 5: Đoạn code nào dưới đây phù hợp cho function predict

- a) `return x*w + b`
- b) `return wx + b`
- c) `return x*w`
- d) `return x.dot(w)+b`

Question 6: Đoạn code nào dưới đây phù hợp cho function gradient

```

a)
loss = y_hat-y
dw = x.T.dot(loss)/len(y)
db = np.sum(loss)/len(y)
cost = np.sum(loss*2)/(2*len(y))

return (db, dw, cost)

b)
loss = y_hat-y
dw = x.T.dot(loss)/len(y)

```

```

db = np.sum(loss)/len(y)
cost = np.sum(loss*2)/(2*len(y))

return (dw,db, cost)

c)
dw = 2*x*(y_hat-y)
db = 2*(y_hat-y)

return (dw, db)

d)
dw = 2*x*(y_hat+y)
db = 2*(y_hat-y)

return (dw, db)

```

Question 7: Đoạn code nào dưới đây phù hợp cho function `update_weight`

```

a)
w_new = w - lr x dw
b_new = b - lr x db

return (w_new, b_new)

b)
w = w - lr*dw
b = b - lr*db

return (w, b)

c)
w_new = w - lr*dw
b_new = b - lr*db

return (w, b)

d)
w_new = w - dw*lr
b_new = b - db*lr

return (w_new, b_new)

```

5. Chuẩn hóa dữ liệu và chia tập train và test:

```

1 from sklearn.preprocessing import StandardScaler
2
3 scalar = StandardScaler()
4
5 ## Your Code here ##
6
7 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
    random_state=42, shuffle=True)

```

6. Huấn luyện mô hình Linear Regression:

Question 8: Đoạn code nào dưới đây phù hợp cho việc huấn luyện mô hình bên dưới sử dụng Batch Gradient Descent

```

1 def linear_regression_vectorized(X, y, learning_rate=0.01, num_iterations=200):
2     n_samples, n_features = X.shape
3     w = np.zeros(n_features) # Initialize weights
4     b = 0 # Initialize bias
5     losses = []
6
7     ##### Your code here #####
8
9     return w, b, losses
10
11 w, b, losses = linear_regression_vectorized(X_train.values, y_train.values,
12                                             learning_rate=0.01, num_iterations=200)
13
14 plt.plot(losses)
15 plt.xlabel('Iteration')
16 plt.ylabel('Loss')
17 plt.title('Loss')
18 plt.show()

```

```

a)
for i in range(N):
    # get a sample
    x = X_train[i]
    y = Y_train[i]

    # predict y_hat
    y_hat = predict(x, w, b)

    # compute loss
    loss = (y_hat-y)*(y_hat-y) / 2.0

    # compute gradient
    (dw, db) = gradient(y_hat, y, x)

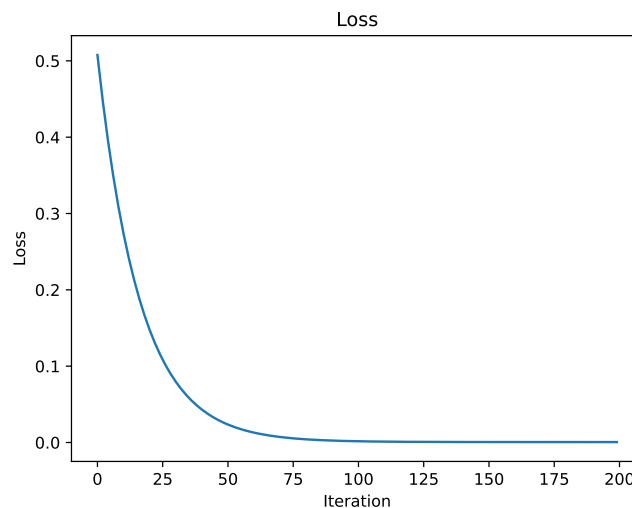
    # update weights
    (w, b) = update_weight(w, b, lr, dw, db)

b)
for _ in range(num_iterations):
    y_hat = predict(X, w, b)
    dw, db, cost = gradient(y_hat, y, X)
    w, b = update_weight(w, b, learning_rate, db, dw)
    losses.append(cost)

c)
for _ in range(num_iterations):
    y_hat = predict(X, w, b)
    dw, db, cost = gradient(y_hat, y, X)
    w, b = update_weight(w, b, learning_rate, dw, db)
    losses.append(cost)

d)
y = predict(X_train, w, b)
db, dw, cost = gradient(y, y_train, X_train)
w, b = update_weight(w, b, lr, dw, db)

```



Hình 6: Biểu đồ hàm loss trong quá trình huấn luyện

7. **Đánh giá mô hình Linear Regression:** Dựa vào thống số Root Mean Square Error (RMSE), Mean Absolute Error (MAE) và R-Square

```

1 from sklearn.metrics import r2_score
2
3 # Make predictions on the test set
4 y_pred = predict(X_test, w, b)
5
6 # Calculate RMSE
7 rmse = np.sqrt(np.mean((y_pred - y_test) ** 2))
8
9 # Calculate MAE
10 mae = np.mean(np.abs(y_pred - y_test))
11
12 # Calculate MAPE
13 mape = np.mean(np.abs((y_test - y_pred) / y_test)) * 100
14
15
16 # Calculate R-squared on training data
17 y_train_pred = predict(X_train, w, b)
18 train_accuracy = r2_score(y_train, y_train_pred)
19
20 # Calculate R-squared on testing data
21 test_accuracy = r2_score(y_test, y_pred)
22
23 print("Root Mean Square Error (RMSE):", round(rmse, 4))
24 print("Mean Absolute Error (MAE):", round(mae, 4))
25 print("Training Accuracy (R-squared):", round(train_accuracy, 4))
26 print("Testing Accuracy (R-squared):", round(test_accuracy, 4))

```

Question 9: Giá trị của RMSE, MAE và R^2 lần lượt là (làm tròn 4 số thập phân):

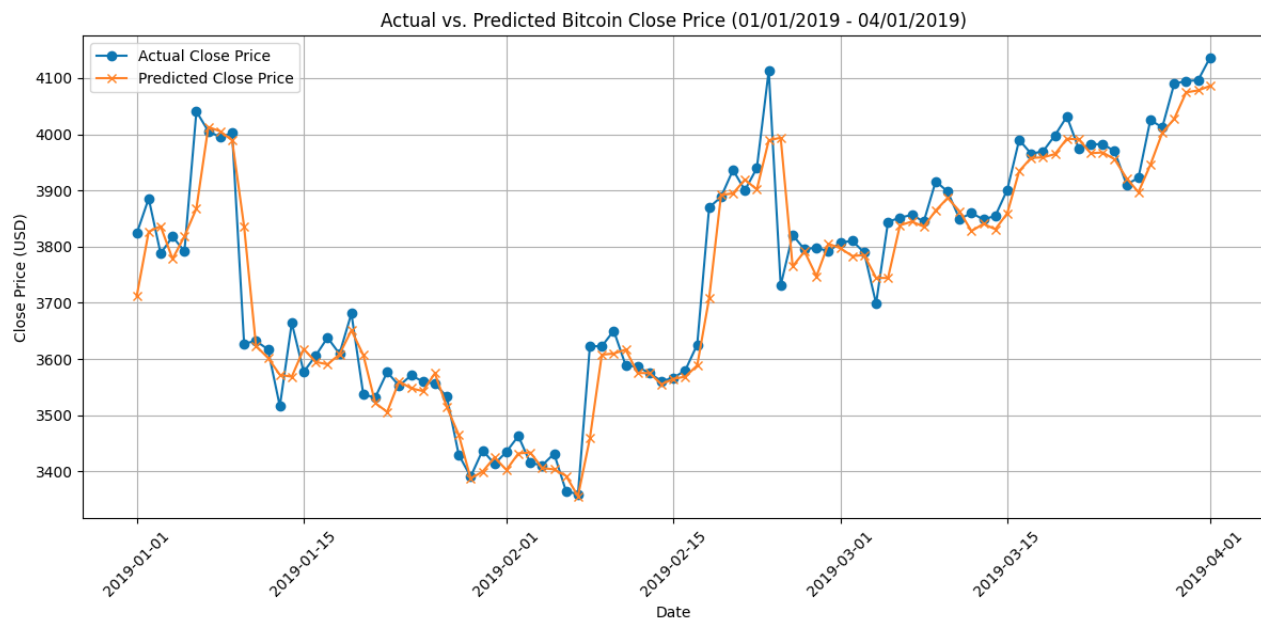
- a) 0.0293 - 0.0138 - 0.9989 - 0.9991
- b) 0.0138 - 0.0293 - 0.9989 - 0.9991
- c) 0.9989 - 0.9991 - 0.0293 - 0.0138
- d) 0.0293 - 0.9991 - 0.9991 - 0.0293

8. Inference: Thực nghiệm mô hình trên giá từ 01/01/2019 - 01/04/2019

```

1 # Filter data for 2019-01-01 to 2019-04-01
2 df_2019_q1 = df[(df['date'] >= '2019-01-01') & (df['date'] <= '2019-04-01')]
3
4 # Prepare X and y for prediction
5 X_2019_q1 = df_2019_q1[["open", "high", "low"]]
6 y_2019_q1_actual = df_2019_q1["close"]
7
8 y_2019_q1_pred = predict(X_2019_q1, w, b)
9
10 # Create the plot
11 plt.figure(figsize=(12, 6))
12 plt.plot(df_2019_q1['date'], y_2019_q1_actual, label='Actual Close Price', marker='o')
13 plt.plot(df_2019_q1['date'], y_2019_q1_pred, label='Predicted Close Price', marker='x')
14 plt.title('Actual vs. Predicted Bitcoin Close Price (01/01/2019 - 04/01/2019)')
15 plt.xlabel('Date')
16 plt.ylabel('Close Price (USD)')
17 plt.legend()
18 plt.grid(True)
19 plt.xticks(rotation=45)
20 plt.tight_layout()
21 plt.show()

```



Hình 7: Biểu đồ so sánh giá kết thúc phiên thực tế và dự đoán của Bitcoin (01/01/2019 - 01/04/2019).