AI VIET NAM – COURSE 2024

Module 4 - Exercise 2

Linear Regression

Ngày 13 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 regresion 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:

```
# dataset
data = genfromtxt('advertising.csv', delimiter=',', skip_header=1)
N = data.shape[0]
X = data[:,:3]
y = data[:,3:]

# Normalize input data by using mean normalizaton
def mean_normalization(X):
  N = len(X)
  maxi = np.max(X)
  mini = np.min(X)
  avg = np.mean(X)
  X = (X-avg) / (maxi-mini)
  X_b = np.c_[np.ones((N, 1)), X]
  return X_b, maxi, mini, avg

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 doán Sales dựa vào các yêu cầu sau:

1. Hoàn thành fucntion **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

```
    def stochastic_gradient_descent(X_b, y, n_epochs=50, learning_rate=0.00001):

    # thetas = np.random.randn(4, 1) # uncomment this line for real application
    thetas = np.asarray([[1.16270837], [-0.81960489],  [1.39501033],
    [0.29763545]])

    thetas_path = [thetas]
    losses = []

    for epoch in range(n_epochs):
        for i in range(N):
            # select random number in N
            # random_index = np.random.randint(N) #In real application, you
    should use this code
            random_index = i # This code is used for this assignment only

            xi = X_b[random_index:random_index+1]
            yi = y[random_index:random_index+1]

            # Compute output
            *******Your code here ********

            # Compute loss li
            *******Your code here ********

            # Compute gradient for loss
            *******Your code here ********

            # Compute gradient
            *******Your code here ********

            # update theta
             *******Your code here ********

            # logging
             *******Your code here ********

    return thetas_path, losses

```

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

```
 sgd_theta, losses = stochastic_gradient_descent(X_b, y, n_epochs=50,
    learning_rate=0.01 )

x_axis = list(range(500))
plt.plot(x_axis,losses[:500], color="r")
plt.show()
```

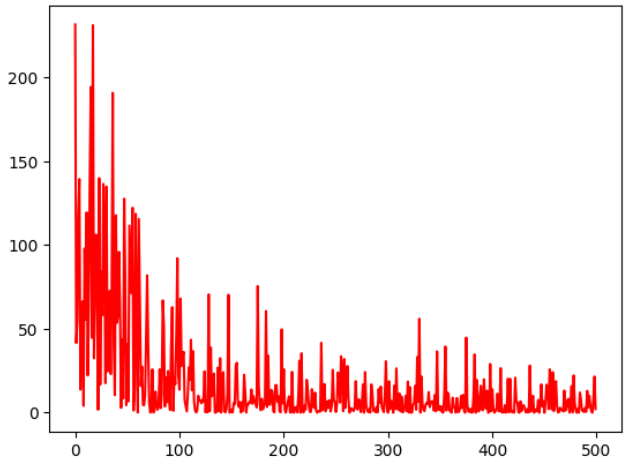
Question 1: Kết quả của đoạn code sau đây:

```
sgd_theta, losses = stochastic_gradient_descent(X_b, y, n_epochs=1, learning_rate
    =0.01 )
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 fucntion **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.

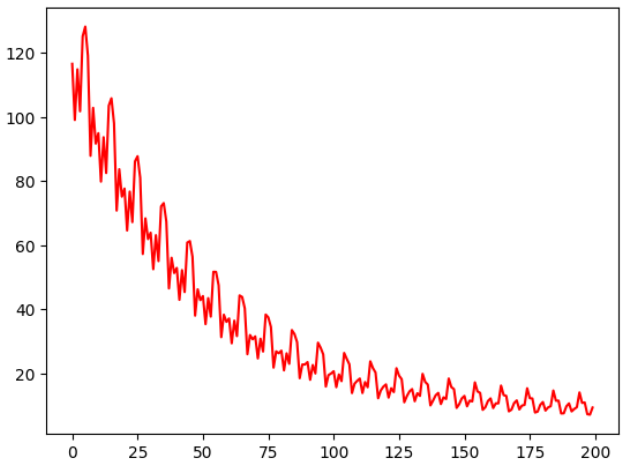
   - **input**: (5 inputs) X_b, y, n_epochs, minibatch_size, learning_rate
   - **output**: thetas_path, losses

```
    def mini_batch_gradient_descent(X_b, y, n_epochs=50,  minibatch_size =
20, learning_rate=0.01):

    # thetas = np.random.randn(4, 1)
    thetas = np.asarray([[1.16270837], [-0.81960489],  [1.39501033],
[0.29763545]])

    thetas_path = [thetas]
    losses = []

    for epoch in range(n_epochs):
        # shuffled_indices = np.random.permutation(N) # uncomment this code for
real application

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


```

```
        X_b_shuffled = X_b[shuffled_indices]
        y_shuffled = y[shuffled_indices]

        for i in range(0, N, minibatch_size):
            xi = X_b_shuffled[i:i+minibatch_size]
            yi = y_shuffled[i:i+minibatch_size]

            # compute output
            *******Your code here ********

            # compute loss
           *******Your code here ********

            # compute derivative of loss
            *******Your code here ********

            # compute derivative of parameters
            *******Your code here ********

            # update parameters
            thetas = thetas - learning_rate*gradients
            thetas_path.append(thetas)

            loss_mean = np.sum(loss)/minibatch_size
            losses.append(loss_mean)

    return thetas_path, losses

```

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

```
mbgd_thetas, losses = mini_batch_gradient_descent(X_b, y, n_epochs=50,
    minibatch_size = 20, learning_rate=0.01)

x_axis = list(range(200))
plt.plot(x_axis,losses[:200], color="r")
plt.show()
```



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

Question 2: Kết quả của đoạn code sau đây:

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

a) 7865.65

b) 6865.65

c) 5865.65

d) 8865.65

3. Hoàn thành fucntion **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.

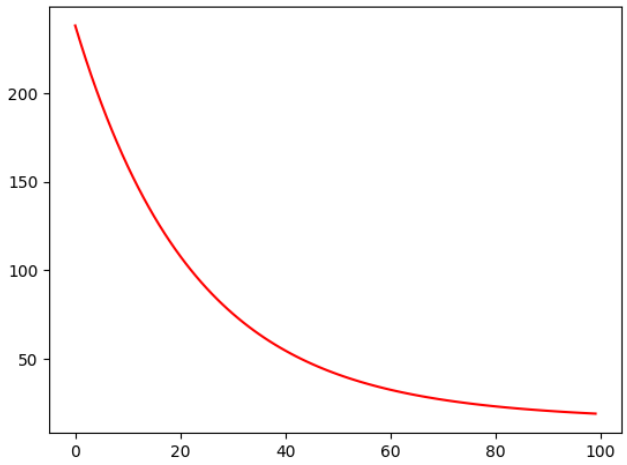
   - **input**: (4 inputs) X_b, y, n_epochs, learning_rate
   - **output**: thetas_path, losses

```
def batch_gradient_descent(X_b, y, n_epochs=100, learning_rate=0.01):

    # thetas = np.random.randn(4, 1) # uncomment this line for real application
    thetas = np.asarray([[1.16270837], [-0.81960489],  [1.39501033],
    [0.29763545]])

    thetas_path = [thetas]
    losses = []

    for i in range(n_epochs):
        # compute output
        *******Your code here ********


        # Compute loss
        *******Your code here ********


        # Compute losss's derivative
        *******Your code here ********


        # computer parameters' derivative
        *******Your code here ********

        # Update parameters
        thetas = thetas - learning_rate*gradients
        thetas_path.append(thetas)

        mean_loss = np.sum(loss)/N
        losses.append(mean_loss)

    return thetas_path, losses
```

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

```
bgd_thetas, losses = batch_gradient_descent(X_b, y, n_epochs=100, learning_rate
    =0.01)

x_axis = list(range(100))
```

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



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

Question 3: Kết quả của đoạn code sau đây:

```
bgd_thetas, losses = batch_gradient_descent(X_b, y, n_epochs=100, learning_rate
    =0.01)
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 momg 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.

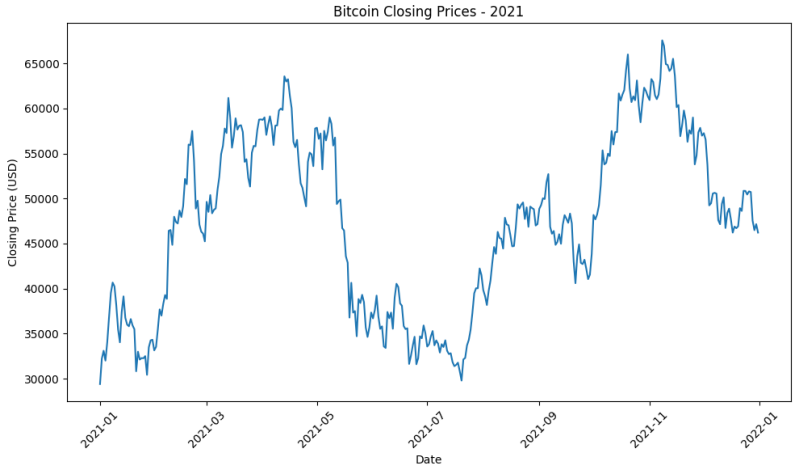
2. **Đọc bộ dữ liệu:** Sử dụng thư viện pandas, chúng ta sẽ đọc file .csv lên như sau:

```
# Load dataset
import pandas as pd

df = pd.read_csv('./BTC-Daily.csv')

# Remove duplicate rows
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 ?

```
# Range of dates covered
df['date'] = pd.to_datetime(df['date'])
date_range = str(df['date'].dt.date.min()) + ' to ' +str(df['date'].dt.date.max())
print(date_range)

######## Your code here ########

for year in unique_years:
    ######## Your code here ########

    merged_data = pd.merge(year_month_day, df, on=['year', 'month', 'day'], how='left')
    # Plot
    plt.figure(figsize=(10, 6))
    plt.plot(merged_data['date_x'], merged_data['close'])
    plt.title(f'Bitcoin Closing Prices - {year}')
    plt.xlabel('Date')
    plt.ylabel('Closing Price (USD)')
    plt.xticks(rotation=45)
    plt.tight_layout()
    plt.show()
```



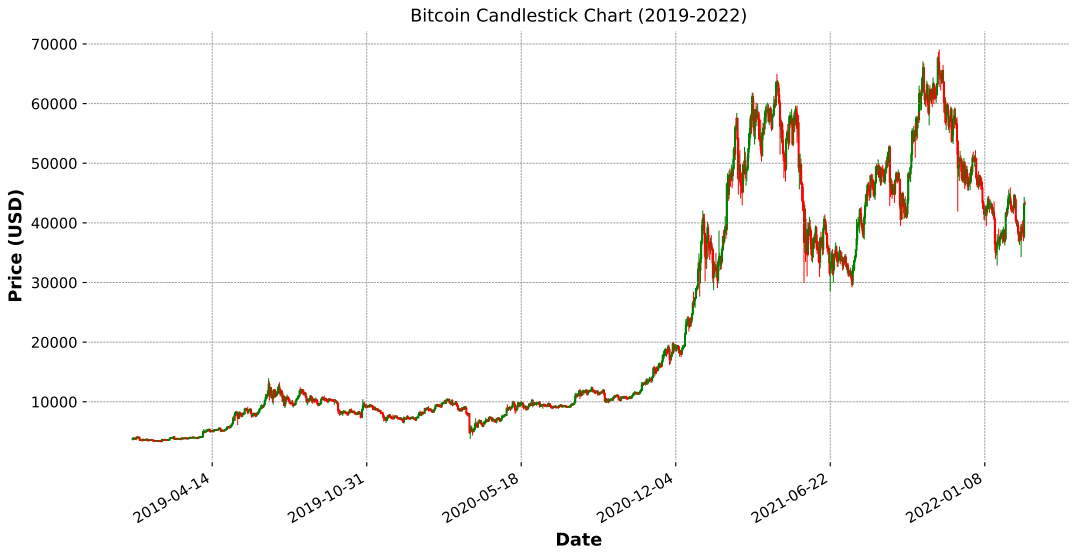
Hình 4: Biểu đồ giá kết thúc phiên trong năm 2021

```
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

```
!pip install mplfinance
```

```
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
from mplfinance.original_flavor import candlestick_ohlc
import datetime

# Filter data for 2019-2022
df_filtered = df[(df['date'] >= '2019-01-01') & (df['date'] <= '2022-12-31')]

# Convert date to matplotlib format
df_filtered['date'] = df_filtered['date'].map(mdates.date2num)

# Create the candlestick chart
fig, ax = plt.subplots(figsize=(20, 6))

candlestick_ohlc(ax, df_filtered[['date', 'open', 'high', 'low', 'close']].values
    , width=0.6, colorup='g', colordown='r')

ax.xaxis.set_major_formatter(mdates.DateFormatter('%Y-%m-%d'))
fig.autofmt_xdate()

plt.title('Bitcoin Candlestick Chart (2019-2022)')
plt.xlabel('Date')
plt.ylabel('Price (USD)')
plt.grid(True)

# Save the plot as a PDF
plt.savefig('bitcoin_candlestick_2019_2022.pdf')

plt.show()
```



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

4. **Tạo mô hình Linear Regression:**

```
def predict(X, w, b):
    ###### Your code here #####

def gradient(y_hat, y, x):
    ###### Your code here #####
    return (dw, db, cost)

def update_weight(w,b,lr,dw,db):
    ###### Your code here #####
    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:**

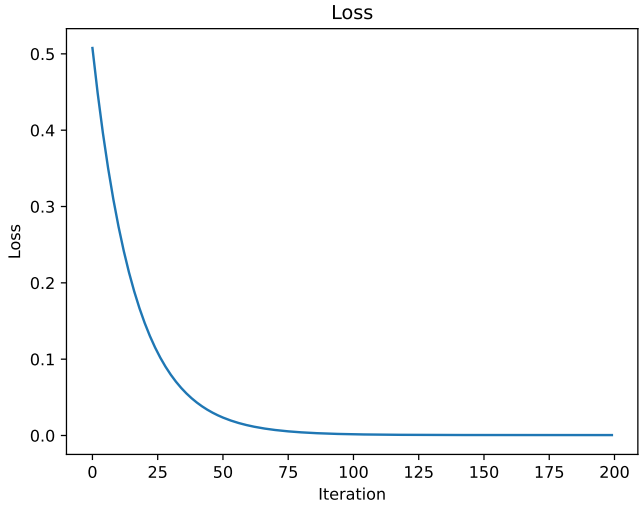
```
from sklearn.preprocessing import StandardScaler

scalar = StandardScaler()

## Your Code here ##

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

```python
def linear_regression_vectorized(X, y, learning_rate=0.01, num_iterations=200):
    n_samples, n_features = X.shape
    w = np.zeros(n_features)  # Initialize weights
    b = 0  # Initialize bias
    losses = []

    ###### Your code here #####

    return w, b, losses

w, b, losses = linear_regression_vectorized(X_train.values, y_train.values,
    learning_rate=0.01, num_iterations=200)


plt.plot(losses)
plt.xlabel('Iteration')
plt.ylabel('Loss')
plt.title('Loss')
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

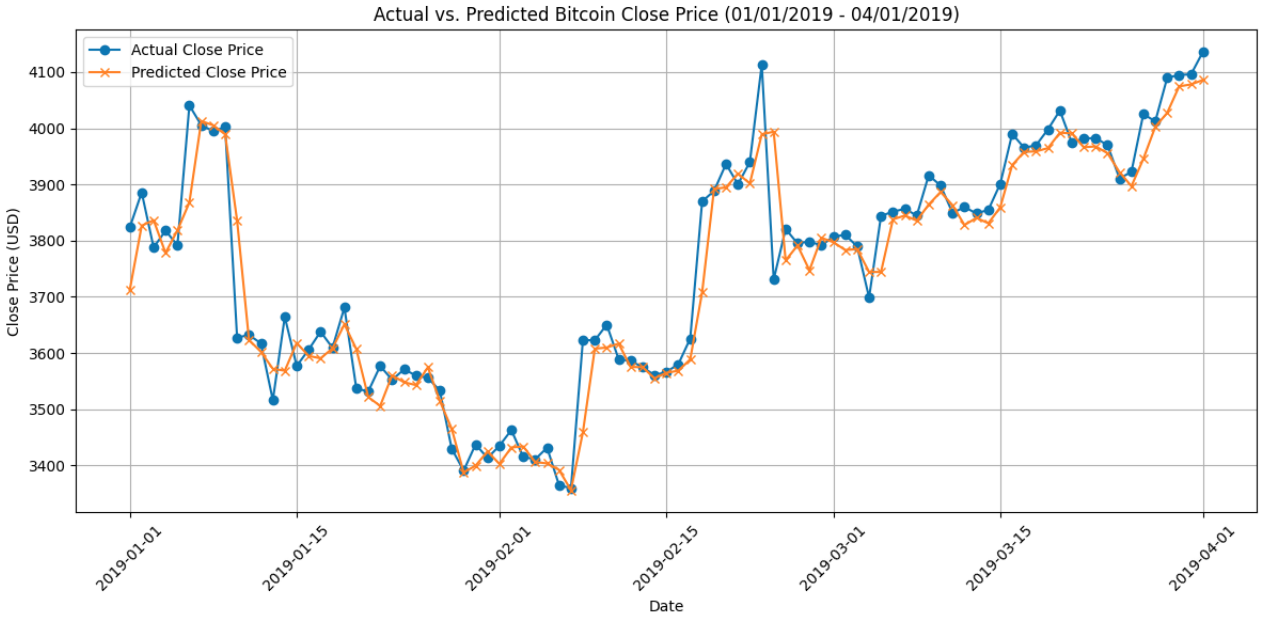
```python
from sklearn.metrics import r2_score

# Make predictions on the test set
y_pred = predict(X_test, w, b)

# Calculate RMSE
rmse = np.sqrt(np.mean((y_pred - y_test) ** 2))

# Calculate MAE
mae = np.mean(np.abs(y_pred - y_test))

# Calculate MAPE
mape = np.mean(np.abs((y_test - y_pred) / y_test)) * 100


# Calculate R-squared on training data
y_train_pred = predict(X_train, w, b)
train_accuracy = r2_score(y_train, y_train_pred)

# Calculate R-squared on testing data
test_accuracy = r2_score(y_test, y_pred)

print("Root Mean Square Error (RMSE):", round(rmse, 4))
print("Mean Absolute Error (MAE):", round(mae, 4))
print("Training Accuracy (R-squared):", round(train_accuracy, 4))
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**

```
# Filter data for 2019-01-01 to 2019-04-01
df_2019_q1 = df[(df['date'] >= '2019-01-01') & (df['date'] <= '2019-04-01')]

# Prepare X and y for prediction
X_2019_q1 = df_2019_q1[["open", "high", "low"]]
y_2019_q1_actual = df_2019_q1["close"]

y_2019_q1_pred = predict(X_2019_q1, w, b)

# Create the plot
plt.figure(figsize=(12, 6))
plt.plot(df_2019_q1['date'], y_2019_q1_actual, label='Actual Close Price', marker
    ='o')
plt.plot(df_2019_q1['date'], y_2019_q1_pred, label='Predicted Close Price',
    marker='x')
plt.title('Actual vs. Predicted Bitcoin Close Price (01/01/2019 - 04/01/2019)')
plt.xlabel('Date')
plt.ylabel('Close Price (USD)')
plt.legend()
plt.grid(True)
plt.xticks(rotation=45)
plt.tight_layout()
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).