* Name: Vo Hong Quan
* ID: 22134012
* Class: 22134

Exercise 1: Linear regression

* Nhập dữ liệu vào và visualize dữ liệu ra, lưu trữ x và y thành ma trận tương ứng

import pandas as pd

import plotly.graph\_objects as go

import numpy as np

pd\_dataframe = pd.read\_csv('ex1.csv')  # Read a CSV file into a DataFrame

x\_col = pd\_dataframe['x']  # Extract the 'x' column

y\_col = pd\_dataframe['y']  # Extract the 'y' column

fig = go.Figure()  # Create a new figure

fig.add\_trace(go.Scatter(x=x\_col,  # Add a scatter plot trace

                         y=y\_col,

                         mode='markers',  # Use marker style for points

                         marker=dict(symbol='x')))  # Set marker style

fig.update\_xaxes(title='x')  # Label for the x-axis

fig.update\_yaxes(title='y', tickangle=0)  # Label for the y-axis and set tick angle

fig.update\_layout(width=600, height=400,  # Update layout with width and height

                  title\_text='Visualize data of ex1.csv file',  # Set plot title

                  showlegend=False)  # Hide legend

fig.show()  # Display the plot

x\_col = pd\_dataframe['x'].values.reshape(-1, 1)  # Reshape 'x' to a column vector

ones\_array = np.ones\_like(x\_col).reshape(-1, 1)  # Create an array of ones for the intercept term

x\_col = np.hstack((ones\_array, x\_col))  # Combine ones array with 'x' values to form [1, x]

y\_col = pd\_dataframe['y'].values.reshape(-1, 1)  # Reshape 'y' to a column vector

A graph with blue dots

Description automatically generated

* Viết hàm tính cost J ở mỗi vòng lặp:

def compute\_cost\_function(\*, H: np.ndarray, Y: np.ndarray) -> np.ndarray:

    m = len(H)  # Number of training examples

    E = H - Y  # Error between predicted and actual values

    # Calculate the cost function J using mean squared error

    J = np.dot(E.T, E) / (2 \* m)

    return J[0, 0]  # Return the scalar cost value

* Viết hàm cập nhật giá trị theta 0 và theta 1 sử dụng phương pháp gradient descent với số lượng vòng lặp và learning rate (alpha) tùy chọn.

def predict(\*, theta: np.array, X: np.array) -> np.ndarray:

    # Compute the hypothesis (predicted values) as the dot product of X and theta

    H = np.matmul(X, theta)

    return H

def update(\*, theta: np.ndarray, X: np.ndarray, H: np.ndarray, Y: np.ndarray, learning\_rate: float) -> np.ndarray:

    E = H - Y  # Error between predicted and actual values

    m = len(X)  # Number of training examples

    # Update theta using the gradient descent formula

    theta\_updated = theta - (learning\_rate / m) \* np.matmul(X.T, E)

    return theta\_updated

def update(\*, theta: np.ndarray, X: np.ndarray, H: np.ndarray, Y: np.ndarray, learning\_rate: float) -> np.ndarray:

    E = H - Y  # Error between predicted and actual values

    m = len(X)  # Number of training examples

    # Update theta using the gradient descent formula

    theta\_updated = theta - (learning\_rate / m) \* np.matmul(X.T, E)

    return theta\_updated

def update\_params(\*, iter: int, theta: np.ndarray, X: np.ndarray, Y: np.ndarray, learning\_rate: float):

    J = []  # Initialize a list to store the cost for each iteration

    for i in range(iter):

        # Predict the output using the current parameters

        H = predict(theta=theta, X=x\_col)

        # Compute and store the cost function value

        J.append(compute\_cost\_function(H=H, Y=y\_col))

        # Update the parameters using gradient descent

        theta = update(theta=theta, X=x\_col, H=predict(theta=theta, X=x\_col), Y=y\_col, learning\_rate=learning\_rate)

    return theta, J  # Return the final parameters and the cost history

theta = np.array([[np.random.random()], [np.random.random()]])  # Initialize random parameters for linear regression

# Initialize the model and perform parameter updates

theta\_updated, J = update\_params(iter=1000, theta=theta, X=x\_col, Y=y\_col, learning\_rate=0.0001)

* Đánh giá tác động của số lượng vòng lặp và các giá trị của learning rate vào hàm cost. Có nghĩa là các bạn phải vẽ sự thay đổi của J sau mỗi vòng lặp ở các giá trị learning rate nhất định.

# Define different learning rates to experiment with during gradient descent

learning\_ = [0.000001, 0.000001, 0.0000025, 0.0000034]

# Generate a range of values from 1 to 1000 to represent the number of iterations for plotting

x\_range = np.linspace(1, 1000, 1000)

# Initialize an array to store the cost values (J) for each learning rate across 1000 iterations

list\_J = np.zeros((1000, 4))

# Create a new Plotly figure for visualizing the cost function over iterations for different learning rates

fig = go.Figure()

# Iterate over the list of learning rates to compute and plot the cost function for each one

for i, value in enumerate(learning\_):

    # Update parameters using gradient descent for each learning rate and store the resulting cost values

    \_, J\_ = update\_params(iter=1000, theta=theta, X=x\_col, Y=y\_col, learning\_rate=value)

    # Store the computed cost values for this learning rate in the corresponding column of list\_J

    list\_J[:, i] = J\_

    # Add a line plot to the figure for this learning rate, showing the cost function over iterations

    fig.add\_trace(

        go.Scatter(

            x=x\_range,  # X-axis: Number of iterations

            y=list\_J[:, i],  # Y-axis: Cost function values

            mode='lines',  # Plot as a line

            name=f'Regression Line {i} - Learning rate: {value}'  # Label for the legend

        )

    )

# Update the X-axis title of the plot

fig.update\_xaxes(title='x')

# Update the Y-axis title and set the tick angle to 0 for better readability

fig.update\_yaxes(title='y', tickangle=0)

# Update the layout of the plot with a title, size, and legend

fig.update\_layout(

    width=1000,  # Set the width of the plot

    height=500,  # Set the height of the plot

    title\_text='Visualize data of ex1.csv file',  # Title of the plot

    showlegend=True  # Show legend

)

# Display the figure with the cost function plots

fig.show()

A graph with a line graph

Description automatically generated with medium confidence

* Biểu diễn đường thẳng cuối cùng tìm thấy và các điểm dữ liệu trên cùng một trục tọa độ

# Generate a range of x values from the minimum to the maximum of the x\_col to plot the regression line

x\_range = np.linspace(x\_col.min(), x\_col.max(), 100)

# Compute the corresponding y values for the regression line using the updated parameters (theta)

# The equation of the line is y = theta\_0 + theta\_1 \* x

y\_range = theta\_updated[0] + theta\_updated[1] \* x\_range

# Create a new Plotly figure to visualize the data points and the regression line

fig = go.Figure()

# Add a scatter plot for the original data points from the CSV file

fig.add\_trace(

    go.Scatter(

        x=pd\_dataframe['x'],  # X values from the 'x' column

        y=pd\_dataframe['y'],  # Y values from the 'y' column

        mode='markers',  # Plot style as markers (points)

        marker=dict(symbol='x'),  # Marker style

        name='Data Points'  # Name for the legend

    )

)

# Add a line plot for the regression line calculated from the model

fig.add\_trace(

    go.Scatter(

        x=x\_range,  # X values for the regression line

        y=y\_range,  # Y values for the regression line

        mode='lines',  # Plot style as a line

        name='Regression Line'  # Name for the legend

    )

)

# Update the X-axis title of the plot

fig.update\_xaxes(title='x')

# Update the Y-axis title and set the tick angle to 0 for better readability

fig.update\_yaxes(title='y', tickangle=0)

# Update the layout of the plot with a title, size, and legend

fig.update\_layout(

    width=800,  # Set the width of the plot

    height=500,  # Set the height of the plot

    title\_text='Visualize data of ex1.csv file',  # Title of the plot

    showlegend=True  # Show legend

)

# Display the figure with data points and the regression line

fig.show()

A graph with a red line

Description automatically generated