ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA KHOA HỌC - KỸ THUẬT MÁY TÍNH

MÔN HỌC: Đồ án tổng hợp - hướng Trí tuệ nhân tạo

FINAL PROJECT

HOUSE PRICE PREDICTION

GVHD: Trần Ngọc Bảo Duy
SV thực hiện: Phạm Nguyễn Long Vũ_1915988
Vũ Quốc Huy_1913579
Nguyễn Văn Nhật Tiến_1915486
Trần Mạnh Dũng_1912966

Tp. Hồ Chí Minh, 12/2021

Mục lục

Danh sách hình vẽ

1 Giới thiệu đề tài

1.1 Giới thiệu sơ lược về đề tài trong thực tế

Ngày nay khi xã hội càng phát triển, khoa học công nghệ càng được áp dụng vào nhiều hơn các lĩnh vực trong cuộc sống. Trong đó có lĩnh vực bất động sản. Bài toán được đưa ra là dựa trên dữ liệu thu thập, phân tích các mối quan hệ tương quan giữa biến phụ thuộc giá nhà và các biến độc lập như số phòng ngủ, diện tích, tuổi thọ,... để tạo ra một mô hình học máy. Từ mô hình học máy, dự đoán được giá nhà từ một dữ liệu đầu vào.

1.2 Ý nghĩa khoa học và thực tiễn của bài toán

Phân tích hồi quy (Linear regression) là một phương pháp thống kê quan trọng để phân tích dữ liệu. Nó cho phép xác định và mô tả các mối quan hệ giữa nhiều yếu tố. Nó cũng cho phép xác định các yếu tố nguy cơ có liên quan đến tiên lượng và tính điểm rủi ro cho tiên lượng cá nhân. Việc áp dụng Linear regression cho ta một cách tiếp cận khoa học trong việc dự đoán giá nhà. Và từ kinh nghiệm này, ta có thể xây dựng những mô hình phức tạp hơn trong tương lai để giải quyết những bài toán khác.

1.3 Phạm vi ứng dụng của bài toán

Linear regression không chỉ được ứng dụng trong việc dự đoán giá nhà, mà là công cụ căn bản trong học máy. Ta có thể áp dụng Linear regression trong mọi lĩnh vực cần mô tả mối quan hệ giữa các yếu tố dựa trên một tập dữ liệu xác định, để có thể dự đoán những yếu tố mà ta mong muốn ngoài tập dữ liệu đã cho.

2 Mô hình Linear Regression

2.1 Giới thiệu về Linear Regression

Linear regression (hồi quy tuyến tính) là một dạng mô hình supervised learning đơn giản, được dùng để dự đoán biến phụ thuộc (y) dựa vào giá trị của biến độc lập (x).

Giả sử ta có một biến phụ thuộc y chịu sự ảnh hưởng bởi các biến độc lập x_1, x_2,..., x_n . Khi đó, ta có thể tìm được hàm biểu diễn mối quan hệ y theo x_1, x_2,..., x_n như sau:

$$y = f(\mathbf{x}) = w_1x_1 + w_2x_2 + ... + w_nx_n + w_0 \tag{1}$$

Khi đó mối quan hệ trên là một mối quan hệ tuyến tính (linear) và bài toán đi tìm các hệ số tối ưu w_0, w_1, w_2,..., w_n là bài toán Linear Regression.

2.2 Sai số dự đoán

Để mô hình có thể dự đoán đúng, ta cần sai số dự đoán e giữa giá trị thực tế y và giá trị dự đoán là thấp nhất:

$$\frac{1}{2}e^2 = \frac{1}{2}(y - \hat{y})^2 = \frac{1}{2}(y - \overline{\mathbf{x}}\mathbf{w})^2 \tag{2}$$

Trong đó $\hat{y}$ là giá trị dự đoán, $\overline{x}$ là vector hàng các giá trị đầu vào của biến độc lập, w là vector cột các hệ số cần tối ưu, hệ số $1/2$ được thêm vào để thuận tiện cho việc lấy đạo hàm.

2.3 Hàm mất mát

Sai số dự đoán xảy ra với tất cả các cặp biến độc lập x và biến phụ thuộc y. Điều ta muốn làm là giảm sai số đến mức nhỏ nhất, nghĩa là tìm w để hàm số sau đạt giá trị nhỏ nhất:

$$\mathcal{L}(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{N}(y_i - \overline{\mathbf{x}}_\mathbf{i}\mathbf{w})^2 \tag{3}$$

Hàm số trên gọi là hàm mất mát (loss function) của bài toán Linear Regression. Ta cần tìm giá trị w sao cho hàm mất mát là nhỏ nhất. Nhưng trước hết, ta sẽ đơn giản hóa hàm mất mát như sau. Đặt Y=[y_1;y_2;....;y_n] là vector cột chứa các output của training data set, $\overline{X} = [\overline{x_1};\overline{x_2};...;\overline{x_n}]$ là ma trận dữ liệu đầu vào với mỗi hàng là một điểm dữ liệu. Khi đó ta có thể tính hàm mất mát như sau:

$$\mathcal{L}(\mathbf{w}) = \frac{1}{2}\|\mathbf{Y} - \overline{\mathbf{X}}\mathbf{w}\|_2^2 \tag{4}$$

Với $\|\mathbf{z}\|_2^2$ là tổng bình phương mỗi phần tử của vector z.

3 Giải thuật Gradient Descent

Giải thuật Gradient Descent là một phương pháp phổ biến để giải bài toán Linear Regression. Phương trình hàm mất mát là:

$$\mathcal{L}(\mathbf{w}) = \frac{1}{2N}\|\mathbf{y} - \overline{\mathbf{X}}\mathbf{w}\|_2^2 \tag{5}$$

Mẫu số có thêm N là số lượng dữ liệu của training data set. Việc lấy giá trị trung bình cộng nhằm giảm khả năng hàm mất mát và đạo hàm là một số rất lớn, ảnh hưởng đến quá trình tính toán. Về mặt toán học, 2 phương trình có nghiệm như nhau. Từ đó ta tính được đạo hàm của hàm mất mát:

$$\nabla_{\mathbf{w}}\mathcal{L}(\mathbf{w}) = \frac{1}{N}\overline{\mathbf{X}}^T(\overline{\mathbf{X}}\mathbf{w} - \mathbf{y}) \tag{6}$$

Ta cần tìm điểm cực tiểu của hàm mất mát $f(\theta)$ với θ là vector tập hợp các tham số của mô hình cần tối ưu. Trong bài toán Linear Regression, θ chính là tham số w. Đạo hàm của hàm số tại điểm θ bất kỳ là $\nabla_\theta f(\theta)$. Ta bắt đầu từ một điểm θ_0 sau đó cứ mỗi vòng lặp ta cập nhật giá trị θ mới theo biểu thức:

$$\theta_{t+1} = \theta_t - \eta\nabla_\theta f\left(\theta_t\right) \tag{7}$$

Trong đó η là *learning rate*. Với giải thuật Gradient Descent, cứ sau mỗi vòng lặp, giá trị đạo hàm càng tiến về 0, giá trị hàm mất mát tiến đến điểm cực tiểu. Để thuật toán có thể chạy hiệu quả, ta cần tìm giá trị learning rate phù hợp. Nếu giá trị này quá nhỏ, sẽ cần số lượng vòng lặp lớn để diễn ra sự hội tụ. Mặt khác, khi learning rate quá to, sự hội tụ diễn ra nhanh nhưng sẽ không đạt được điểm cực tiểu mà cứ quanh quẩn xung quanh điểm cực tiểu.

4 Mô hình dự đoán giá nhà

4.1 Thư viện sử dụng

- *numpy* `import numpy as np`

  NumPy là gói cơ bản cho tính toán khoa học bằng Python. Nó là một thư viện Python cung cấp đối tượng mảng đa chiều, các đối tượng dẫn xuất khác nhau (chẳng hạn như mảng và ma trận có mặt nạ) và một loạt các quy trình cho các hoạt động nhanh trên mảng, bao gồm toán học, logic, thao tác hình dạng, sắp xếp, lựa chọn, I / O , các phép biến đổi Fourier rời rạc, đại số tuyến tính cơ bản, các phép toán thống kê cơ bản, mô phỏng ngẫu nhiên và hơn thế nữa.

- *pandas* `import pandas as pd`

  Pandas là một gói Python mã nguồn mở được sử dụng rộng rãi nhất cho các nhiệm vụ khoa học dữ liệu / phân tích dữ liệu và học máy. Nó được xây dựng dựa trên Numpy, cung cấp hỗ trợ cho các mảng đa chiều. Là một trong những gói bao bọc dữ liệu phổ biến nhất, Pandas hoạt động tốt với nhiều mô-đun khoa học dữ liệu khác bên trong hệ sinh thái Python và thường được bao gồm trong mọi bản phân phối Python, từ những gói đi kèm với hệ điều hành của bạn đến các bản phân phối của nhà cung cấp thương mại như ActiveState's ActivePython.

- *matplotlib.pyplot* `import matplotlib.pyplot as plt`

  matplotlib.pyplot là một tập hợp các hàm làm cho matplotlib hoạt động giống như MATLAB. Mỗi hàm pyplot thực hiện một số thay đổi đối với một hình: ví dụ: tạo một hình, tạo vùng vẽ đồ thị trong một hình, vẽ một số đường trong vùng vẽ đồ thị, trang trí đồ thị bằng nhãn, v.v.

- *sweetviz* `import sweetviz as sv`

  Sweetviz là một thư viện Python mã nguồn mở tạo ra các hình ảnh trực quan mật độ cao, đẹp mắt để khởi động EDA (Phân tích Dữ liệu Khám phá) chỉ với hai dòng mã. Đầu ra là một ứng dụng HTML hoàn toàn độc lập. Hệ thống được xây dựng xung quanh việc nhanh chóng hình dung các giá trị mục tiêu và so sánh các tập dữ liệu.

4.2 Mô hình dự đoán giá nhà sử dụng Linear Regression và Gradient Descent

- Import dataset as dataframe

```
def import_data(path, column_id):
    # Import data set
    df = pd.read_csv(path, header = 0)
    # Drop column id
    df = df.drop(axis = 1, columns = column_id)
    return df
```

- Report dataset with sweetviz

```
def report_data(df, path):
```

```
    report = sv.analyze(df)
    report.show_html(path)
```

- Handle with missing and nan values

```
def handle_with_nan_values(df):
    m = df.shape[0] # pre-number of traing examples
    # MasVnrType, MasVnrArea and Electrical column
    # If nan values occupies less or eual than 1 percent, delete all
nan-rows
    number_nan_value_in_masvnr = df["MasVnrType"].isna().sum()
    percent_nan_value_in_masvnr = (number_nan_value_in_masvnr * 100) // m
    if percent_nan_value_in_masvnr <= 1:
        df.dropna(subset = ["MasVnrType"], inplace = True)
    number_nan_value_in_electrical = df["Electrical"].isna().sum()
    percent_nan_value_in_electrical = (number_nan_value_in_electrical * 100)
// m
    if percent_nan_value_in_electrical <= 1:
        df.dropna(subset = ["Electrical"], inplace = True)
    # Reset index after delete nan-rows
    df.reset_index(drop = True, inplace = True)
    return df
def handle_with_missing_value(df, object_to_fillna):
    # Base on data description to decide
    df.fillna(value = object_to_fillna, inplace = True)
    return df
```

- Handle non-numerical features and normalize numerical features

```
def non_numerical_handle(df):
    columns = df.columns.values
    def convert_to_int(val):
        return text_digit_value[val]
    for column in columns:
        text_digit_value = {}
        if df[column].dtype = "int64" and df[column].dtype = "float64":
            column_list = df[column].tolist()
```

```
            column_set = set(column_list)
            x = 0
            for unique_element in column_set:
               if unique_element not in text_digit_value:
               text_digit_value[unique_element] = x
               x += 1
            df[column] = list(map(convert_to_int, df[column]))
      return df
   def pre_feature_scaling(df):
      columns = df.columns.values
      feature_scaling_parameters = []
      def feature_normalization(val):
         return (val - mean_value) / std_value;
      for column in columns:
         if df[column].dtype == "int64" or df[column].dtype == "float64":
            mean_value = df[column].mean()
            std_value = df[column].std()
            feature_scaling_parameters.append([mean_value, std_value])
            df[column] = df[column].apply(feature_normalization)
      return {"DataFrame": df, "Parameter": feature_scaling_parameters}
   def post_feature_scaling(df, feature_scaling_parameters):
      columns = df.columns.values
      column_index = 0
      def feature_normalization(val):
         return (val - mean_value) / std_value;
      for column in columns:
         if df[column].dtype == "int64" or df[column].dtype == "float64":
            mean_value = feature_scaling_parameters[column_index][0]
            std_value = feature_scaling_parameters[column_index][1]
            cloumn_index += 1
            df[column] = df[column].apply(feature_normalization)
      return df
```

- Gradient Descent

```
## Gradient Descent without Regularization
def computeCost(error_value, m):
    J = np.dot(error_value, np.transpose(error_value))
    J = J / (2 * m)
    return J
def gradientDescent(X, y, theta, m, alpha, num_iters):
    J_history = []
    for i in range(num_iters):
        error_value = np.zeros((1, m))
        error_value += np.concatenate(np.dot(X, theta)) - y
        J_history.append(computeCost(error_value, m))
        theta = theta - (alpha / m) * np.transpose(np.dot(error_value, X))
    return [J_history, theta]
## Gradient Descent with L1 Regularization
def computeCost_Ridge(error_value, m, lambda_parameter, theta):
    J = np.dot(error_value, np.transpose(error_value)) + lambda_parameter *
np.dot(np.transpose(theta), theta)
    J = J / (2 * m)
    return J
def gradientDescent_Ridge(X, y, theta, m, alpha, num_iters,
lambda_parameter):
    J_history = []
    for i in range(num_iters):
        error_value = np.zeros((1, m))
        error_value += np.concatenate(np.dot(X, theta)) - y
        J_history.append(computeCost_Ridge(error_value, m, lambda_parameter,
theta))
        theta = theta - (alpha / m) * (np.transpose(np.dot(error_value, X)) +
lambda_parameter * theta)
    return [J_history, theta]
## Gradient Descent with L2 Regularization
def computeCost_Lasso(error_value, m, lambda_parameter, theta):
```

```
    J = np.dot(error_value, np.transpose(error_value)) + lambda_parameter *
sum(np.abs(theta))
    J = J / (2 * m)
    return J
def gradientDescent_Lasso(X, y, theta, m, alpha, num_iters,
lambda_parameter):
    J_history = []
    for i in range(num_iters):
        error_value = np.zeros((1, m))
        error_value += np.concatenate(np.dot(X, theta)) - y
        J_history.append(computeCost_Lasso(error_value, m, lambda_parameter,
theta))
        theta = theta - (alpha / m) * (np.transpose(np.dot(error_value, X)) +
(lambda_parameter / 2) * np.sign(theta))
    return [J_history, theta]
```

5 Hiện thực bài toán

5.1 Thay thế dữ liệu NA

Dựa vào mô tả của dữ liệu [2] thì ta sẽ quyết định những trường dữ liệu sau khi có giá trị NA sẽ được thay thế.

- LotFrontage: 0,
- Alley: "#None",
- BsmtQual: "#None",
- BsmtCond: "#None",
- BsmtExposure: "#None",
- BsmtFinType1: "#None",
- BsmtFinType2: "#None",
- FireplaceQu: "#None",
- GarageType: "#None",
- GarageYrBlt: 0,
- GarageFinish: "#None",
- GarageQual: "#None",
- GarageCond: "#None",
- PoolQC: "#None",
- Fence: "#None",
- MiscFeature: "#None"

Sau khi quyết định các dữ liệu thì ta lưu toàn bộ định nghĩa trên vào dict `object_to_fillna`

5.2 Nhập dữ liệu

Trên Google Colab, thực hiện thêm các file train.csv và test.csv [2] vào session storage

```
df = import_data(path = "train.csv", column_id = ["Id"])
```

Hoặc upload file file train.csv và test.csv [2] vào Google Drive

```
from google.colab import drive
drive.mount("/content/drive")
df = import_data(path = "/content/drive/.../train.csv", column_id = "Id")
```

5.3 Xử lí tiền dữ liệu

```
df = handle_with_nan_values(df)
y = df["SalePrice"]
```

```
X = df.drop(axis = 1, columns = ["SalePrice"])
X = handle_with_missing_value(X, object_to_fillna)
feature_scaling = pre_feature_scaling(X)
X = feature_scaling["DataFrame"]
X = non_numerical_handle(X)
m = X.shape[0] # pos-number of traing examples
n = X.shape[1] # number of features
X.insert(0, "Theta Zero", np.ones((m, 1), dtype = int), True)
X = X.to_numpy()
y = y.to_numpy()
```

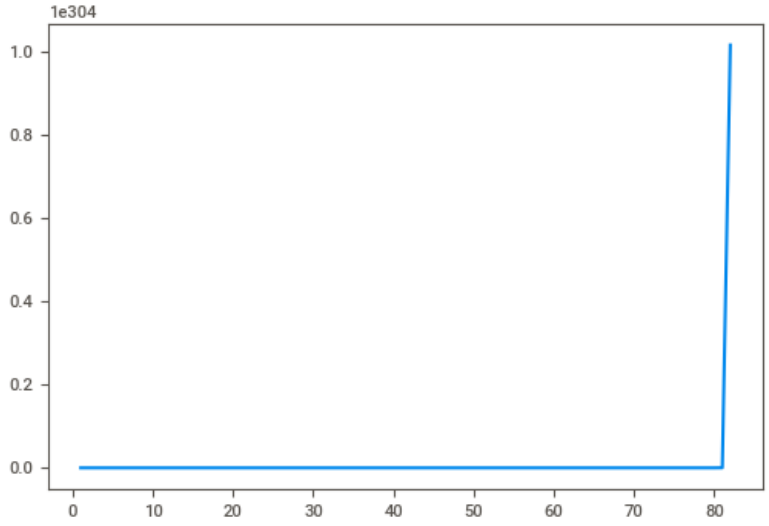
5.4 Lựa chọn tham số

```
alpha = 0.001 # Learning Rate
num_iters = 2000 # Number of iterations
lambda_parameter = 5000 # Regularization Lambda Parameter
```

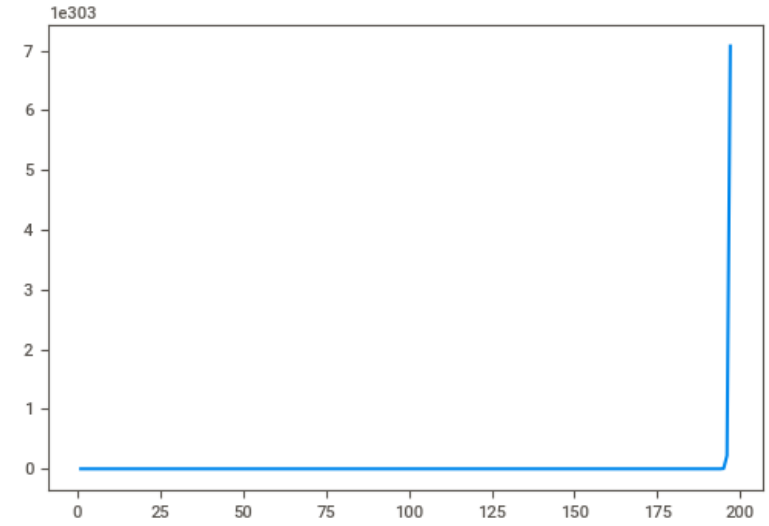
5.5 Thuật toán Gradient Descent

```
# Run Gradient Descent without regularization
theta = np.zeros(((n + 1), 1), dtype = int)
[J_history, theta] = gradientDescent(X, y, theta, m, alpha, num_iters)
J_history = np.concatenate(J_history)
plt.plot(np.arange(1, num_iters + 1), J_history)
```

- Với alpha = 0.1
- Với alpha = 0.01
- Với alpha = 0.001

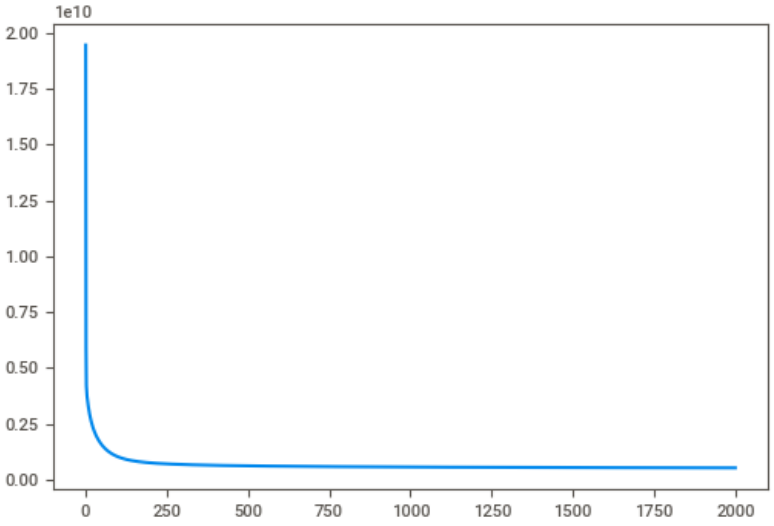


Hình 1: Cost Function with alpha = 0.1



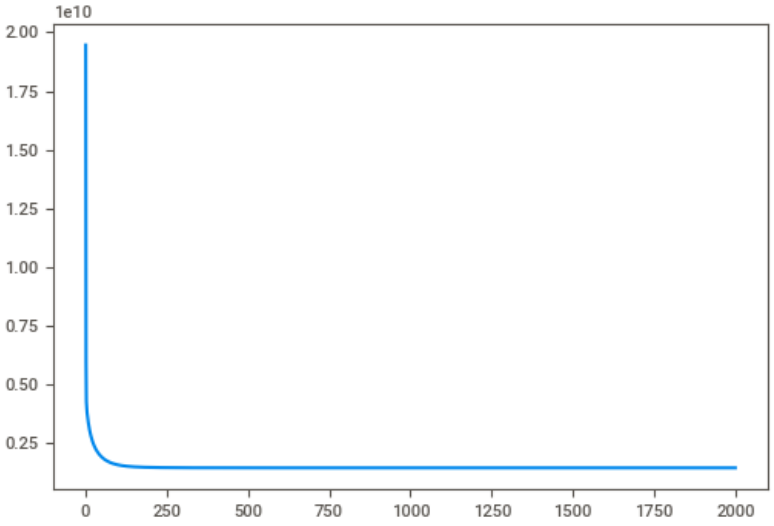
Hình 2: Cost Function with alpha = 0.01

```
# Run Gradient Descent with L1 regularization

theta_Ridge = np.zeros(((n + 1), 1), dtype = int)

[J_history_Ridge, theta_Ridge] = gradientDescent_Ridge(X, y, theta_Ridge, m,
alpha, num_iters, lambda_parameter)
```



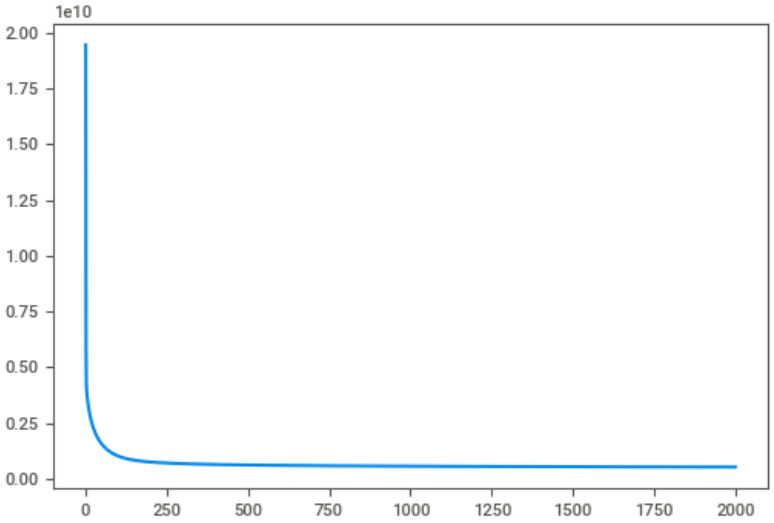
Hình 3: Cost Function with alpha = 0.001

```
J_history_Ridge = np.concatenate(J_history_Ridge)
plt.plot(np.arange(1, num_iters + 1), J_history_Ridge)
```



Hình 4: Cost Function (L1 regularization) with alpha = 0.001

```
# Run Gradient Descent with L2 regularization
theta_Lasso = np.zeros(((n + 1), 1), dtype = int)
[J_history_Lasso, theta_Lasso] = gradientDescent_Lasso(X, y, theta_Lasso, m,
alpha, num_iters, lambda_parameter)
J_history_Lasso = np.concatenate(J_history_Lasso)
plt.plot(np.arange(1, num_iters + 1), J_history_Lasso)
```



Hình 5: Cost Function (L2 regularization) with alpha = 0.001

6 Demo: Thực thi chương trình trên Google Colab

6.1 Nhập dữ liệu cần dự đoán

```
X_test = import_data(path = "/content/drive/.../test.csv", column_id = "Id")
y_test = import_data(path = "/content/drive/.../sample_submission.csv",
column_id = ["Id"])
```

Hoặc,

```
X_test = import_data(path = "test.csv", column_id = "Id") y_test =
import_data(path = "sample_submission.csv", column_id = "Id")
```

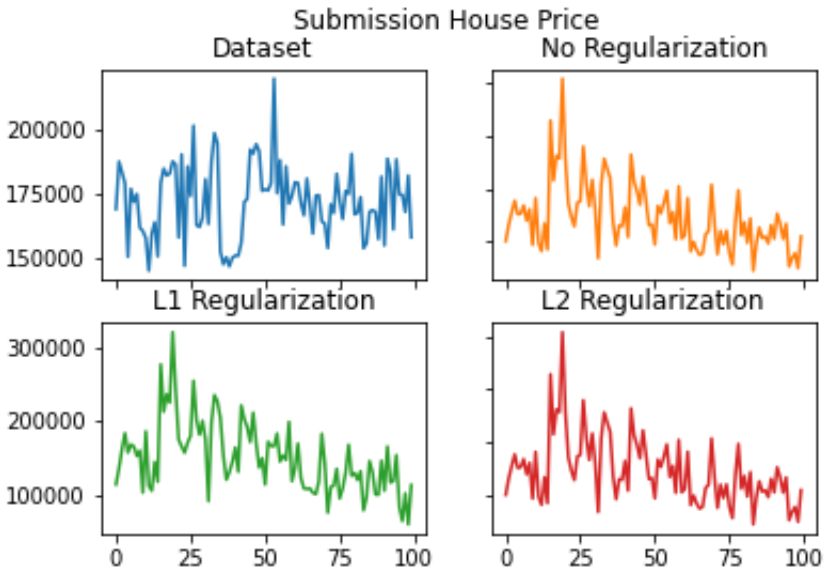
6.2 Xử lí tiền dữ liệu

```
df_test = pd.concat([X_test, y_test], axis = 1) # Pre-processing

df_test.dropna(subset = ["MasVnrType"], inplace = True)

df_test.dropna(subset = ["Electrical"], inplace = True)

df_test.reset_index(drop = True, inplace = True)

y_test = df_test["SalePrice"]

X_test = df_test.drop(axis = 1, columns = ["SalePrice"])

X_test = handle_with_missing_value(X_test, object_to_fillna)

feature_scaling_parameters = feature_scaling["Parameter"]

X_test = post_feature_scaling(X_test, feature_scaling_parameters)

X_test = non_numerical_handle(X_test)

number_of_test_example = X_test.shape[0]

X_test.insert(0, "Theta Zero", np.ones((number_of_test_example, 1), dtype =
int), True)

X_test = X_test.to_numpy()

y_test = y_test.to_numpy()
```

6.3 Dự đoán giá nhà

```
# Predict with normal model

def predict_house_price(test_example, theta):

   return np.dot(test_example, theta)

# Predict with L1 regularization model

def predict_house_price(test_example, theta_Ridge):

   return np.dot(test_example, theta_Ridge)

# Predict with L2 regularization model
```

```
def predict_house_price(test_example, theta_Lasso):
    return np.dot(test_example, theta_Lasso)
```



Hình 6: House Price in the first 100 test examples

7 Đánh giá

7.1 So sánh giá nhà được dự đoán so với giá nhà ban đầu

Hình 7: House Price in the first 100 training examples

Hình 8: House Price in the last 100 training examples

7.2 So sánh giá trị của hàm chi phí của 3 phương án

```
#Compare the value of the cost function of the 3 methods above
print("No Regularization", J_history[-1])
print("Ridge Regularization", J_history_Ridge[-1])
print("Lasso Regularization", J_history_Lasso[-1])
```

Output:

```
No Regularization [5.19745442e+08]
Ridge Regularization [1.57966981e+09]
Lasso Regularization [5.20154045e+08]
```

Ở đây, hàm chi phí của 3 phương án sau khi thực hiện 2000 và với learning rate là 0.001 đều mang giá trị khá lớn (e+08), lí do là vì trong mẫu dữ liệu dataset ở cột giá nhà (y) thì có giá trong đoạn [34900, 755000] và nếu ta thực hiện scale giá trị của cột giá nhà thành [3.49, 75.5] thì giá trị của hàm chi phí chỉ còn

```
No Regularization [5.19745442]
Ridge Regularization [15.796698108]
Lasso Regularization [5.20154045]
```

Hoặc ta có thể thực hiện feature scaling lên cột y và đồng thời lưu trữ giá trị trung bình và độ lệch chuẩn thì ta có thể có được giá trị tốt hơn.

8 Tham khảo

[1] **Ridge and Lasso Regression: L1 and L2 Regularization**. *Saptashwa Bhattacharyya*.

https://towardsdatascience.com/ridge-and-lasso-regression-a-complete-guide-with-python-scikit-learn-e20e34bcbf0b.

[2] **House Prices - Advanced Regression Techniques**.

https://www.kaggle.com/c/house-prices-advanced-regression-techniques.

[3] **Machine learning cơ bản - Bài 3: Linear Regression**.

https://machinelearningcoban.com/2016/12/28/linearregression/.

[4] **Machine learning cơ bản - Bài 7: Gradient Descent**.

https://machinelearningcoban.com/2017/01/12/gradientdescent//.