

VIETNAM GENERAL CONFEDERATION OF LABOUR
TON DUC THANG UNIVERSITY
FACULTY OF INFORMATION TECHNOLOGY



INTRODUCTION TO MACHINE LEARNING

FINAL PROJECT

Instructor: GV LÊ ANH CƯỜNG

Executor: HOÀNG NGỌC NGỌ - 520H0388

Course: 24

HO CHI MINH CITY, 2023

VIETNAM GENERAL CONFEDERATION OF LABOUR
TON DUC THANG UNIVERSITY
FACULTY OF INFORMATION TECHNOLOGY



INTRODUCTION TO MACHINE LEARNING

FINAL PROJECT

Instructor: **GV LÊ ANH CƯỜNG**

Executor: **HOÀNG NGỌC NGỌ - 520H0388**

Course: **24**

HO CHI MINH CITY , 2023

THANK YOU

In this middle exam about Machine Learning subject so we are grateful for teacher's supporting who is Dr. Lê Anh Cường because he helped us to complete the midterm as soon as possible. Thank you very much.

THE PROJECT IS COMPLETED AT TON DUC THANG UNIVERSITY

I hereby declare that this is my own project product and is guided by teacher Le Anh Cuong. The research content and results in this topic are honest and have not been published in any form before. The data in the tables for analysis, comments, and evaluation were collected by the author from different sources and clearly stated in the reference section.

In addition, the project also uses a number of comments, assessments as well as data from other authors and other organizations, all with citations and source notes.

If any fraud is discovered, I will take full responsibility for the content of my project. Ton Duc Thang University is not involved in copyright violations caused by me during the implementation process (if any).

HCM city, date month year

Author

(sign and write full name)

INSTRUCTOR VERIFICATION AND EVALUATION SECTION

Confirmation from the instructor

HCM city, date month year

(sign and write full name)

The teacher's evaluation part marks the test

HCM city, date month year

(sign and write full name)

SUMMARY

TABLE OF CONTENTS

THANK YOU.....	
INSTRUCTOR VERIFICATION AND EVALUATION SECTION.....	
SUMMARY.....	
TABLE OF CONTENTS.....	
QUESTIONS.....	
ANSWER.....	

QUESTIONS

Bài 1 (3 điểm):

Trình bày một bài nghiên cứu, đánh giá của em về các vấn đề sau:

- 1) Tìm hiểu, so sánh các phương pháp Optimizer trong huấn luyện mô hình học máy;
- 2) Tìm hiểu về Continual Learning và Test Production khi xây dựng một giải pháp học máy để giải quyết một bài toán nào đó.

Bài 2: (7 điểm):

Đưa ra một bài toán dự đoán có thể giải quyết bằng học máy (machine learning) với các yêu cầu sau:

- Số Feature/Attribute gồm nhiều kiểu: categorical và numerical;
 - Dữ liệu phải chưa được học, thực tập trên lớp và trong bài tập về nhà;
- 1) Phân tích thống kê trên dữ liệu, vẽ các đồ thị để hiểu bài toán, hiểu dữ liệu. Tìm hiểu các đặc trưng và đánh giá vai trò của các đặc trưng đối với mục tiêu bài toán;
 - 2) Ứng dụng các mô hình học máy cơ bản để giải quyết bài toán, bao gồm cả các mô hình thuộc Ensemble Learning;
 - 3) Sử dụng Feed Forward Neural Network và Recurrent Neural Network (hoặc mô thuộc loại này) để giải quyết bài toán;
 - 4) Áp dụng các kỹ thuật tránh Overfitting trên các mô hình của câu (2) và câu (3) để giải quyết bài toán;
 - 5) Sau khi huấn luyện xong mô hình thì muốn cải thiện độ chính xác, ta sẽ làm gì để giải quyết nó? Phân tích các trường hợp sai, đề ra giải pháp và thực hiện nó, sau đó đánh giá xem có cải tiến so với trước không.

ANSWER

Question 1 :

I. Tìm hiểu, so sánh các phương pháp Optimizer trong huấn luyện mô hình học máy:

- Thuật toán tối ưu là cơ sở để xây dựng mô hình neural network với mục đích "học" được các features (hay pattern) của dữ liệu đầu vào, từ đó có thể tìm 1 cặp weights và bias phù hợp để tối ưu hóa model.

- Các thuật toán tối ưu :

1. Gradient Descent (GD)

- Gradient Descent dịch ra tiếng Việt là giảm dần độ dốc, nên hướng tiếp cận ở đây là chọn 1 nghiệm ngẫu nhiên cứ sau mỗi vòng lặp (hay epoch) thì cho nó tiến dần đến điểm cần tìm.

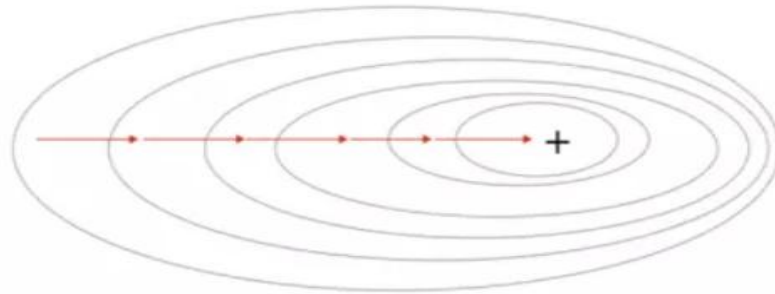
1. Dự đoán một điểm khởi tạo $\theta = \theta_0$.
2. Cập nhật θ đến khi đạt được kết quả chấp nhận được:

$$\theta = \theta - \eta \nabla_{\theta} J(\theta)$$

với $\nabla_{\theta} J(\theta)$ là đạo hàm của hàm mất mát tại θ .

- Gradient descent phụ thuộc vào nhiều yếu tố : như nếu chọn điểm x ban đầu khác nhau sẽ ảnh hưởng đến quá trình hội tụ; hoặc tốc độ học (learning rate) quá lớn hoặc quá nhỏ cũng ảnh hưởng: nếu tốc độ học quá nhỏ thì tốc độ hội tụ rất chậm ảnh hưởng đến quá trình training, còn tốc độ học quá lớn thì tiến nhanh tới đích sau vài vòng lặp tuy nhiên thuật toán không hội tụ, quanh quẩn quanh đích vì bước nhảy quá lớn.

Gradient Descent



- Ưu điểm :

+ Thuật toán gradient descent cơ bản, dễ hiểu. Thuật toán đã giải quyết được vấn đề tối ưu model neural network bằng cách cập nhật trọng số sau mỗi vòng lặp.

- Nhược điểm :

+ Vì đơn giản nên thuật toán Gradient Descent còn nhiều hạn chế như phụ thuộc vào nghiệm khởi tạo ban đầu và learning rate.

+ Ví dụ 1 hàm số có 2 global minimum thì tùy thuộc vào 2 điểm khởi tạo ban đầu sẽ cho ra 2 nghiệm cuối cùng khác nhau.

+ Tốc độ học quá lớn sẽ khiến cho thuật toán không hội tụ, quanh quẩn bên đích vì bước nhảy quá lớn; hoặc tốc độ học nhỏ ảnh hưởng đến tốc độ training.

2. Stochastic Gradient Descent (SGD)

- Stochastic là 1 biến thể của Gradient Descent . Thay vì sau mỗi epoch chúng ta sẽ cập nhật trọng số (Weight) 1 lần thì trong mỗi epoch có N điểm dữ liệu chúng ta sẽ cập nhật trọng số N lần.. Công thức SGD cũng tương tự như GD nhưng thực hiện trên từng điểm dữ liệu.

- Công thức :

$$\theta = \theta - \eta \nabla_{\theta} J(\theta; \mathbf{x}_i; \mathbf{y}_i)$$

trong đó $J(\theta; \mathbf{x}_i; \mathbf{y}_i)$ là hàm mất mát với chỉ 1 cặp điểm dữ liệu (input, label) là $(\mathbf{x}_i, \mathbf{y}_i)$. **Chú ý:** chúng ta hoàn toàn có thể áp dụng các thuật toán tăng tốc GD như Momentum, AdaGrad,... vào SGD.

- Ưu điểm :

+ Thuật toán giải quyết được đối với cơ sở dữ liệu lớn mà GD không làm được. Thuật toán tối ưu này hiện nay vẫn hay được sử dụng.

- Nhược điểm:

+ Thuật toán vẫn chưa giải quyết được 2 nhược điểm lớn của gradient descent (learning rate, điểm dữ liệu ban đầu). Vì vậy ta phải kết hợp SGD với 1 số thuật toán khác như: Momentum, AdaGrad,..

3. Momentum

- Để khắc phục các hạn chế trên của thuật toán Gradient Descent người ta dùng gradient descent with momentum.

- Gradient Descent with Momentum là một biến thể của thuật toán Gradient Descent, trong đó sử dụng một động lượng (momentum) để cập nhật trọng số. Ý tưởng chính là tích tụ một giá trị động lượng từ các bước trước đó và sử dụng nó để điều chỉnh bước cập nhật hiện tại.

- ta có công thức Momentum:

$$v_t = \gamma v_{t-1} + \eta \nabla_{\theta} J(\theta)$$

Trong đó γ thường được chọn là một giá trị khoảng 0.9, v_t là vận tốc tại thời điểm trước đó, $\nabla_{\theta} J(\theta)$ chính là độ dốc của điểm trước đó. Sau đó vị trí mới của *hòn bi* được xác định như sau:

$$\theta = \theta - v_t$$

Thuật toán đơn giản này tỏ ra rất hiệu quả trong các bài toán thực tế (trong không gian nhiều chiều, cách tính toán cũng hoàn toàn tương tự). Dưới đây là một ví dụ trong không gian một chiều.

- Ưu điểm :

+ Thuật toán tối ưu giải quyết được vấn đề: Gradient Descent không tiến được tới điểm global minimum mà chỉ dừng lại ở local minimum.

- Nhược điểm :

+ Khi Gradient Descent with Momentum tiến gần đến điểm tối ưu, *hòn bi* có thể có đà nên vẫn di chuyển qua lại quanh điểm tối ưu thay vì dừng ngay lập tức. Điều này gây ra hiện tượng overshooting, khi *hòn bi* "đi quá" điểm tối ưu và sau đó quay lại, tạo ra một chu kỳ dao động trước khi hội tụ.

4. Adagrad

- Không giống như các thuật toán trước đó thì learning rate hầu như giống nhau trong quá trình training (learning rate là hằng số), Adagrad coi learning rate là 1 tham số. Tức là Adagrad sẽ cho learning rate biến thiên sau mỗi thời điểm t .

- Công thức :

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \epsilon}} \odot g_t$$

Trong đó :

η : hằng số

g_t : gradient tại thời điểm t

ϵ : hệ số tránh lỗi (chia cho mẫu bằng 0)

G : là ma trận chéo mà mỗi phần tử trên đường chéo (i,i) là bình phương của đạo hàm vector tham số tại thời điểm t .

- Ưu điểm:

+ Một lợi ích dễ thấy của Adagrad là tránh việc điều chỉnh learning rate bằng tay, chỉ cần để tốc độ học default là 0.01 thì thuật toán sẽ tự động điều chỉnh.

- Nhược điểm:

+ Yếu điểm của Adagrad là tổng bình phương biến thiên sẽ lớn dần theo thời gian cho đến khi nó làm tốc độ học cực kì nhỏ, làm việc training trở nên đóng băng.

5. RMSprop

- RMSprop giải quyết vấn đề tỷ lệ học giảm dần của Adagrad bằng cách chia tỷ lệ học cho trung bình của bình phương gradient.

- Công thức :

$$E[g^2]_t = 0,9E[g^2]_{t-1} + 0,1g_t^2$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t$$

- Ưu điểm :

+ Ưu điểm rõ nhất của RMSprop là giải quyết được vấn đề tốc độ học giảm dần của Adagrad (vấn đề tốc độ học giảm dần theo thời gian sẽ khiến việc training chậm dần, có thể dẫn tới bị đóng băng)

- Nhược điểm :

+ Thuật toán RMSprop có thể cho kết quả nghiệm chỉ là local minimum chứ không đạt được global minimum như Momentum. Vì vậy người ta sẽ kết hợp cả 2 thuật toán Momentum với RMSprop cho ra 1 thuật toán tối ưu Adam. Chúng ta sẽ trình bày nó trong phần sau.

6. Adam

- Adam được xem như là sự kết hợp của RMSprop và Stochastic Gradient Descent với động lượng. Adam là một phương pháp tỉ lệ học thích ứng, nó tính toán tỉ lệ học tập cá nhân cho các tham số khác nhau. Adam sử dụng ước tính của khoảng thời gian thứ nhất và thứ hai của độ dốc để điều chỉnh tỉ lệ học cho từng trọng số của mạng nơ-ron.

- Tuy nhiên, qua nghiên cứu thực nghiệm, trong một số trường hợp, Adam vẫn còn gặp phải nhiều thiếu sót so với thuật toán SGD. Thuật toán Adam được mô tả:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

Trong đó, v_t là trung bình động của bình phương và m_t là trung bình động của gradient; β_1 và β_2 là tốc độ của di chuyển.

II. Tìm hiểu về Continual Learning và Test Production khi xây dựng một giải pháp học máy để giải quyết một bài toán nào đó.

1. Continual Learning

a. Continual Learning là gì ?

- Continual Learning được xây dựng dựa trên ý tưởng học tập liên tục về thế giới bên ngoài nhằm tạo điều kiện cho sự phát triển tự chủ, gia tăng các kỹ năng và kiến thức phức tạp hơn bao giờ hết.

- Continual Learning là một thuật toán thích ứng có khả năng học hỏi từ một luồng thông tin liên tục, với thông tin đó sẽ dần dần có sẵn theo thời gian và khi số lượng nhiệm vụ cần thực hiện được học (ví dụ: các lớp thành viên trong một nhiệm vụ phân loại) không được xác định trước. Điều quan trọng là việc cung cấp thông tin mới phải diễn ra mà không gây lãng quên hoặc can thiệp nghiêm trọng.

- Thường được mô tả một cách không chính thức bằng tập hợp mong muốn sau đây dành cho hệ thống tính toán:

- i. Học hỏi dần dần từ các luồng dữ liệu
- ii. Thể hiện sự truyền tải thông tin tiến lùi theo thời gian.
- iii. Tránh sự lãng quên nghiêm trọng dữ liệu trước đó

iv. Thích ứng với những thay đổi trong phân phối dữ liệu

b. The Catastrophic Forgetting phenomenon

- Một vấn đề lớn của của Continual Learning environment là The Catastrophic Forgetting phenomenon có thể tóm tắt ngắn gọn trong một câu: . Sự quên lãng thảm khốc (hoặc đơn giản là quên) là vấn đề chính mà các thuật toán Học liên tục phải đối mặt. quá trình học kiến thức mới nhanh chóng phá vỡ những thông tin đã thu được trước đó.

- Thật không may, tất cả mô hình kết nối đều phải chịu sự Lãng quên thảm khốc. Hậu quả là mạng lưới thần kinh không phù hợp để học trong môi trường Học tập liên tục vì hiệu suất của chúng trong các nhiệm vụ trước đó sẽ suy giảm rất nhanh.

- Sự quên lãng thảm khốc có thể được đặc trưng bằng cách xem xét tình thế tiến thoái lưỡng nan sự ổn định-dẻo dai: một mô hình học tập phải đủ linh hoạt để học thông tin mới, nhưng nó cũng phải ổn định để bảo tồn tri thức bên trong. Sự đánh đổi này không bao giờ được thỏa mãn đối với các mạng thần kinh truyền thống, nơi mà tính linh hoạt dễ dàng lấn át sự ổn định.

c. Beyond forgetting

- Beyond forgetting là trọng tâm chính của Học tập liên tục, vẫn có những khía cạnh khác cần được xem xét khi học liên tục. Việc bảo tồn kiến thức cũ không chỉ quan trọng để thực hiện tốt các nhiệm vụ trước đó. Nó cũng có thể được sử dụng để thực hiện tốt hơn các nhiệm vụ sắp tới. Tính năng này, được gọi là transfer learning , cho phép thuật toán Continual Learning chỉ yêu cầu một vài ví dụ về nhiệm vụ mới để thành thạo.

2. Test Production

1. Test production là gì ?

- Là việc thực hiện kiểm tra các mô hình học máy đang được sử dụng trong môi trường sản xuất. Mục đích của Test Production là đảm bảo rằng các mô hình học máy vẫn hoạt động chính xác và hiệu quả trong môi trường thực tế.

- Khi xây dựng một giải pháp học máy để giải quyết một bài toán nào đó, các nhà phát triển cần chú ý đến Test Production ngay từ giai đoạn đầu của quá trình phát triển. Điều này sẽ

giúp đảm bảo rằng các mô hình học máy được triển khai trong môi trường sản xuất có chất lượng cao và đáp ứng được các yêu cầu của bài toán.

2. Phương pháp Test Production

- *Test thủ công*: Test thủ công được thực hiện bởi các kỹ sư hoặc chuyên gia kiểm thử. Phương pháp này thường được sử dụng để kiểm tra các trường hợp edge case hoặc các trường hợp phức tạp.

- *Test tự động*: Test tự động được thực hiện bằng cách sử dụng các công cụ và framework chuyên dụng. Phương pháp này giúp tiết kiệm thời gian và công sức cho các nhà phát triển.

- *Test dựa trên dữ liệu*: Test dựa trên dữ liệu sử dụng dữ liệu thực tế từ môi trường sản xuất để kiểm tra các mô hình học máy. Phương pháp này giúp đảm bảo rằng các mô hình học máy có thể xử lý các dữ liệu thực tế một cách chính xác.

Question 2

1. Mô tả bài toán

- Lấy bối cảnh ở nước Mỹ, chúng ta sẽ đóng vai trò như một đại lý nhà nước để dự đoán giá nhà cho các khu vực. Với tập dữ liệu đã được chuẩn bị sẵn, nhiệm vụ bây giờ đó là sử dụng mô hình hồi quy tuyến tính để có thể ước tính ngôi nhà sẽ được bán với giá bao nhiêu.

- Tập tài liệu được lấy tại nguồn Kaggle.

- Trong tập dữ liệu này có 7 cột và 5000 hàng :

- Avg.Area Income : Thu nhập trung bình tại khu vực ngôi nhà đã bán
- Avg.Area House Age : Trung bình tuổi của một ngôi nhà
- Avg. Area Number of Rooms : Trung bình diện tích các phòng
- Avg. Area Number of Bedrooms : Trung bình diện tích các phòng
- Area Population : Dân số tại khu vực bán nhà
- Price: Giá ngôi nhà đã bán

- Address: Địa chỉ ngôi nhà đã bán

- Xem tổng quát data này ta có thể sử dụng

Data columns (total 7 columns):

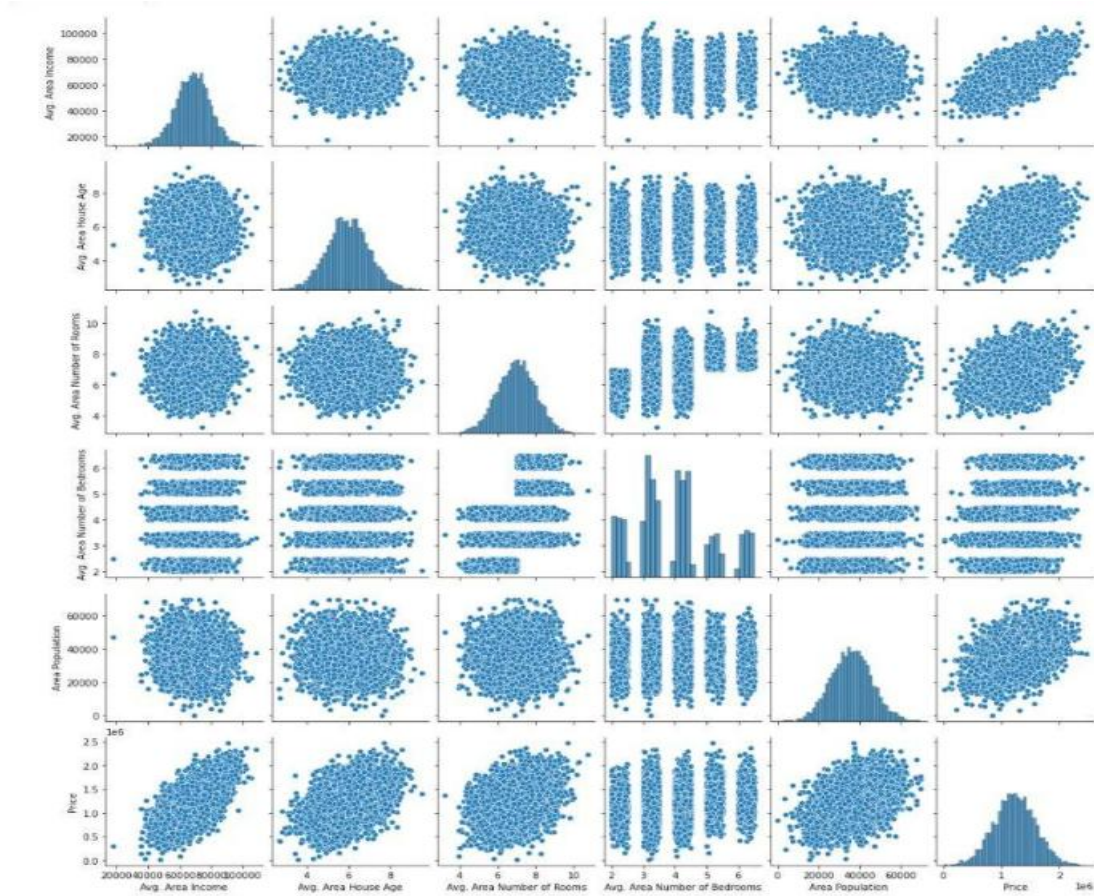
```
#      Column                                     Non-Null Count  Dtype
---  -
0      Avg. Area Income                           5000 non-null   float64
1      Avg. Area House Age                         5000 non-null   float64
2      Avg. Area Number of Rooms                    5000 non-null   float64
3      Avg. Area Number of Bedrooms                 5000 non-null   float64
4      Area Population                              5000 non-null   float64
5      Price                                          5000 non-null   float64
6      Address                                       5000 non-null   object
dtypes: float64(6), object(1)
memory usage: 273.6+ KB
```

- Ta có thể thống kê dữ liệu bằng describe():

	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population	Price
count	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5.000000e+03
mean	68583.108984	5.977222	6.987792	3.981330	36163.516039	1.232073e+06
std	10657.991214	0.991456	1.005833	1.234137	9925.650114	3.531176e+05
min	17796.631190	2.644304	3.236194	2.000000	172.610686	1.593866e+04
25%	61480.562388	5.322283	6.299250	3.140000	29403.928702	9.975771e+05
50%	68804.286404	5.970429	7.002902	4.050000	36199.406689	1.232669e+06
75%	75783.338666	6.650808	7.665871	4.490000	42861.290769	1.471210e+06
max	107701.748378	9.519088	10.759588	6.500000	69621.713378	2.469066e+06

2. EDA

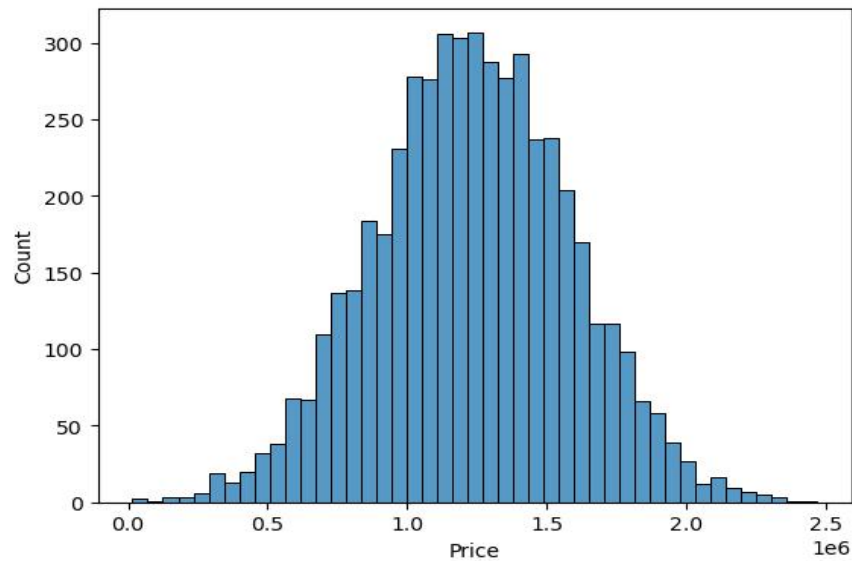
- Phân bố bằng Seaborn ta có thể thấy được tương quan về dữ liệu



- Về tương quan giữa các cột, ta thấy Cột Price có kiểu phân tán theo mô hình tuyến tính, dựa trên thông tin này, ta xây dựng mô hình máy học hồi quy tuyến tính để dự đoán nó dựa trên giá trị các cột khác, trừ cột địa chỉ (Address) ngôi nhà.

- Bây giờ chúng ta sẽ sử dụng `histplot()` để vẽ biểu đồ giá nhà :

```
sns.histplot(x = data['Price']);
```

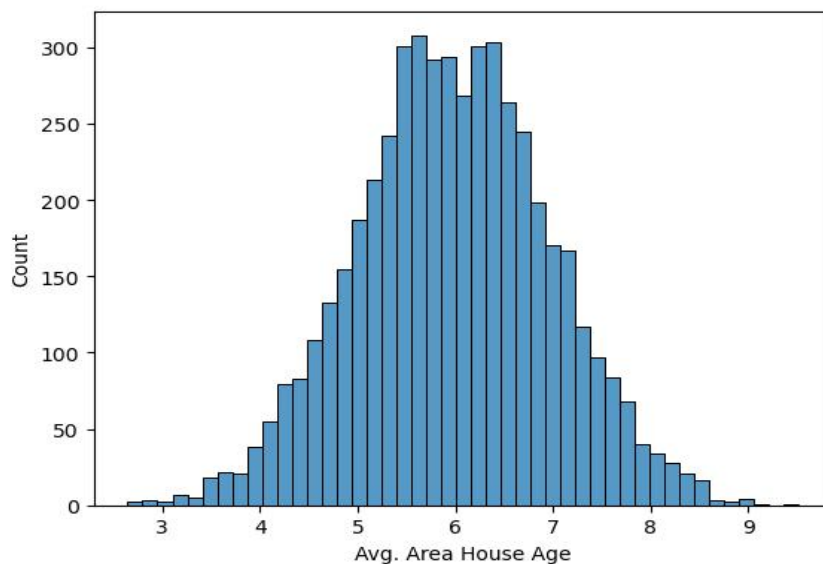


- Ta thấy giá các ngôi nhà đã bán thường tập trung ở mức giá 0.5 đến 2.0, và nhiều nhất là 0.8 đến 1.7.

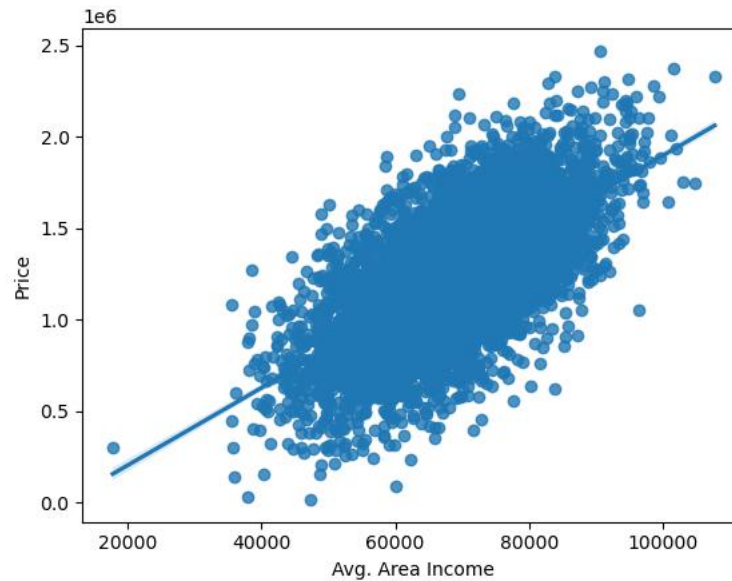
- Bây giờ chúng ta sẽ sử dụng `histplot()` để vẽ biểu đồ thu thập trung bình tại khu vực ngôi nhà đã bán

```
sns.histplot(x = data['Avg. Area House Age'])
```

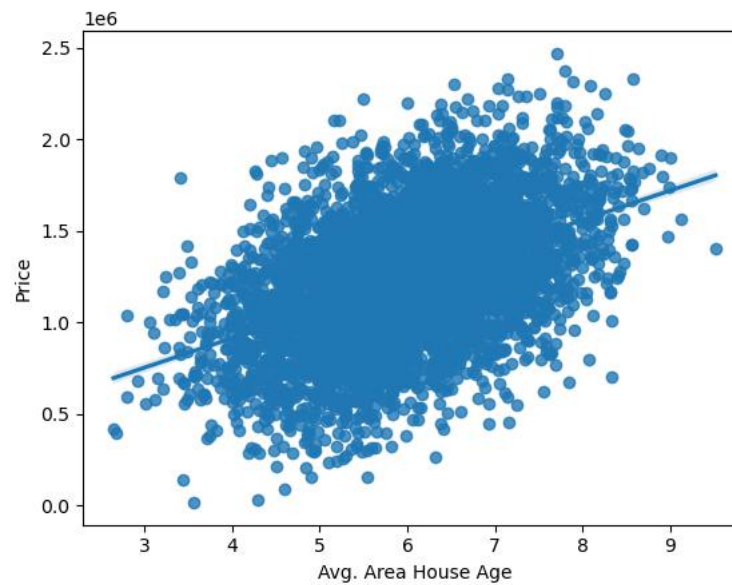
<Axes: xlabel='Avg. Area House Age', ylabel='Count'>



- Bây giờ chúng ta sẽ sử dụng `regplot()` để vẽ biểu đồ phân tán cho thấy mối quan hệ giữa thu nhập bình quân khu vực và giá nhà.

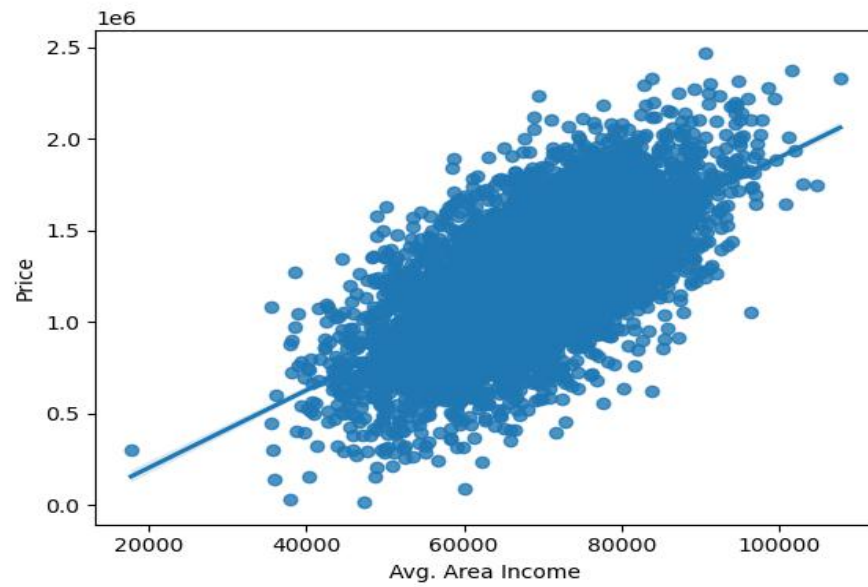


- Biểu đồ phân tán cho thấy mối quan hệ giữa trung bình tuổi của một ngôi nhà và giá nhà.



- Biểu đồ phân tán cho thấy mối quan hệ giữa thu thập trung bình tại khu vực ngôi nhà đã bán và giá nhà.

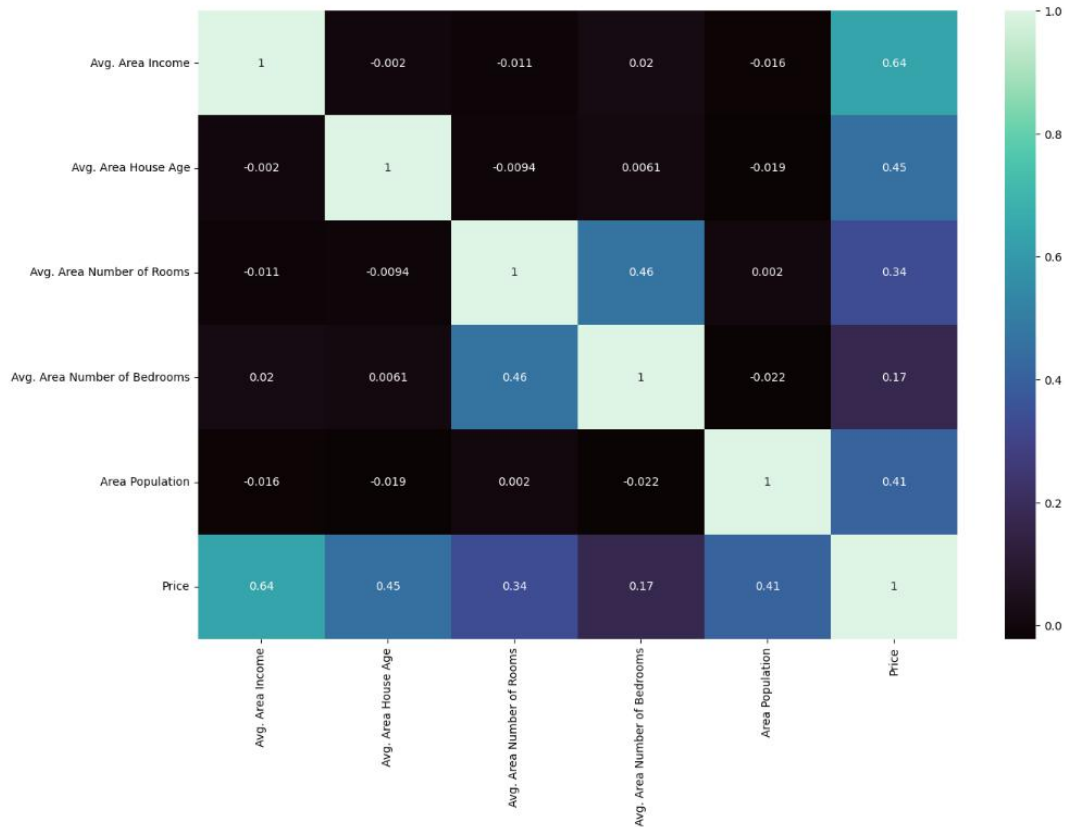
```
|: sns.regplot(x = data['Avg. Area Income'], y = data['Price']);
```



- Chúng ta sẽ sử dụng bản đồ nhiệt để kiểm tra độ tương quan giữa các cột:

```
[16]: data=data.drop('Address',axis=1)
plt.figure(figsize=(15, 10))
sns.heatmap(data.corr(), annot=True, cmap='mako')
```

[16]: <Axes: >



Qua đó, ta phân tích được các cột có giá trị tương quan như thế nào với nhau. Về cơ bản, cột giá (price) có chút tương quan với các cột còn lại nhiều nhất, chứng tỏ các yếu tố đó có tác động ít nhiều lên giá nhà

3. Xây dựng các học máy Linear Regression, Decision Trees, Random forest,

- Chuẩn bị dữ liệu :

Bây giờ chúng ta hãy bắt đầu đào tạo mô hình hồi quy. Trước tiên, chúng ta sẽ cần tách dữ liệu của mình thành một mảng X chứa các tính năng cần đào tạo (các biến độc lập) và một mảng y với biến mục tiêu (biến phụ thuộc), trong trường hợp này là cột Giá. Chúng ta sẽ loại bỏ cột Địa chỉ vì nó chỉ có thông tin văn bản mà mô hình hồi quy tuyến tính không thể sử dụng

```

X = data[['Avg. Area Income', 'Avg. Area House Age', 'Avg. Area Number of Rooms',
          'Avg. Area Number of Bedrooms', 'Area Population']]
y = data['Price']

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline

pipeline = Pipeline([
    ('std_scaler', StandardScaler())
])
X_train = pipeline.fit_transform(X_train)
X_test = pipeline.transform(X_test)

X_train

array([[ -0.19049241, -0.12817719, -0.13160635,  0.12038585, -0.82761782],
       [-1.38876401,  0.43080443,  0.80028487, -0.55648895,  1.15829878],
       [-0.35012392,  0.46680752,  1.70375078,  0.03067955, -0.31904298],
       ...,
       [-0.22335061,  0.53809182, -0.36489661, -0.68697084,  0.11908894],
       [-0.92417067,  1.43077434,  2.26846315,  0.2753331 ,  1.39018355],
       [-0.69357335, -0.07762332,  0.89219611,  1.67801341, -0.00681852]])

X_test

array([[ -0.62396497,  1.05134233, -0.53493732, -0.59726454,  0.77509854],
       [-1.06752524,  0.92593776, -0.05804915, -0.69512596,  0.73880748],
       [ 0.14995479,  0.77674776, -0.31465336, -1.60849918, -0.69076777],
       ...,
       [ 1.16115701,  1.18417775,  0.25662849,  1.18870632,  0.93991305],
       [ 1.69832503,  0.56046124, -1.85607396, -1.54325824,  1.16585758],
       [-0.14145775, -1.04516314,  0.64633545,  0.8625016 , -0.40292631]])

```

1. LINEAR REGRESSION

Code

```

# Linear Regression
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.model_selection import GridSearchCV, cross_val_score
from statistics import mean
# Xây dựng mô hình hồi quy tuyến tính
model = LinearRegression()
# Lắp mô hình vào dữ liệu huấn luyện
model.fit(X_train, y_train)

```

▼ LinearRegression
LinearRegression()

```
[ ] model.score(X_test, y_test)
```

```
0.917997170683436
```

```

# Model Evaluation
y_pred = model.predict(X_test)

```

```

▶ accuracies=cross_val_score(estimator = model, X = X_train, y = y_train, cv = 10)
print( accuracies)
print('Mean score: %0.3f'% accuracies.mean())
print('-----')

from sklearn import metrics
r_squared = metrics.r2_score(y_test, y_pred)
print('MAE:', metrics.mean_absolute_error(y_test, y_pred))
print('MSE:', metrics.mean_squared_error(y_test, y_pred))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test,y_pred)))
print('R-squared:', r_squared)

```

Kết quả đánh giá mô hình :

```

▶ accuracies=cross_val_score(estimator = model, X = X_train, y = y_train, cv = 10)
print( accuracies)
print('Mean score: %0.3f'% accuracies.mean())
print('-----')

from sklearn import metrics
r_squared = metrics.r2_score(y_test, y_pred)
print('MAE:', metrics.mean_absolute_error(y_test, y_pred))
print('MSE:', metrics.mean_squared_error(y_test, y_pred))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test,y_pred)))
print('R-squared:', r_squared)

[0.90917181 0.91727003 0.91281918 0.90697909 0.91750831 0.92150226
 0.92029966 0.92025849 0.91942055 0.92761298]
Mean score: 0.917
-----
MAE: 80879.09723489163
MSE: 10089009300.893644
RMSE: 100444.0605555831
R-squared: 0.917997170683436

```

2. DECISION TREES

Code:


```

# Decision Trees
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import GridSearchCV, cross_val_score
x = data.drop(['Price'], axis = 1)
y = data.Price
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Training
Model2 = DecisionTreeRegressor(random_state = 42)
Model2.fit(X_train, y_train)

# Prediction
y_pred2 = Model2.predict(X_test)

# Accuracy
accuracies = cross_val_score(estimator = DTree_reg, X = X_train, y = y_train, cv = 10)
print( accuracies)
print('Mean score: %0.3f'% accuracies.mean())
print('-----')

# Evaluation
from sklearn import metrics
r2_DTree = r2_score(y_test, y_pred2)
mae_DTree = metrics.mean_absolute_error(y_test, y_pred2)
mse_DTree = metrics.mean_squared_error(y_test, y_pred2)
rmse_DTree = np.sqrt(metrics.mean_squared_error(y_test, y_pred2))
print(' R2      :%0.2f' % r2_DTree)
print(' MAE      :%0.2f' % mae_DTree)
print(' MSE      :%0.2f' % mse_DTree)
print(' RMSE      :%0.2f' % rmse_DTree)

```

Kết quả đánh giá mô hình :

```

[0.73766102 0.7509694 0.75010489 0.71140884 0.76288635 0.7481817
 0.74721003 0.75191648 0.72351055 0.78886533]
Mean score: 0.747

```

```

-----
R2      :0.74
MAE      :140823.96
MSE      :31568012111.93
RMSE      :177673.89

```

3. KNN

#Code


```
[ ] # KNeighborsRegressor
from sklearn.neighbors import KNeighborsRegressor
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
Model4=KNeighborsRegressor(n_neighbors=1)
Model4.fit(X_train,y_train)
```

▼ KNeighborsRegressor
KNeighborsRegressor(n_neighbors=1)

```
[ ] # Prediction
y_pred4=Model4.predict(X_test)
```

```
▶ accuracies=cross_val_score(estimator = model, X = X_train, y = y_train, cv = 10)
print( accuracies)
print('Mean score: %0.3f'% accuracies.mean())
print('-----')
from sklearn import metrics
r2 = r2_score(y_test,y_pred4)
mae = metrics.mean_absolute_error(y_test,y_pred4)
mse = metrics.mean_squared_error(y_test, y_pred4)
rmse = np.sqrt(metrics.mean_squared_error(y_test,y_pred4))
print(' R2      :%0.2f' % r2)
print(' MAE      :%0.2f' % mae)
print(' MSE      :%0.2f' % mse)
print(' RMSE     :%0.2f' % rmse)
```

Kết quả đánh giá mô hình :

```
👤 [0.90917181 0.91727003 0.91281918 0.90697909 0.91750831 0.92150226
    0.92029966 0.92025849 0.91942055 0.92761298]
Mean score: 0.917
-----
R2      :0.17
MAE      :252262.24
MSE      :101590678265.35
RMSE     :318732.93
```

4. RANDOM FOREST

Code

```
[ ] #RANDOM FOREST

from sklearn.ensemble import RandomForestRegressor
Model3=RandomForestRegressor()

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
Model3.fit(X_train,y_train)
```

▼ RandomForestRegressor
RandomForestRegressor()

```
[ ] #Prediction
y_pred3=Model3.predict(X_test)
```

```
▶ accuracies=cross_val_score(estimator = model, X = X_train, y = y_train, cv = 10)
print( accuracies)
print('Mean score: %0.3f'% accuracies.mean())
print('-----')
from sklearn import metrics
r2 = r2_score(y_test,y_pred3)
mae = metrics.mean_absolute_error(y_test,y_pred3)
mse = metrics.mean_squared_error(y_test, y_pred3)
rmse = np.sqrt(metrics.mean_squared_error(y_test,y_pred3))
print(' R2      :%0.2f' % r2)
print(' MAE      :%0.2f' % mae)
print(' MSE      :%0.2f' % mse)
print(' RMSE     :%0.2f' % rmse)
```

Kết quả đánh giá mô hình :

```
▶ [0.90917181 0.91727003 0.91281918 0.90697909 0.91750831 0.92150226
 0.92029966 0.92025849 0.91942055 0.92761298]
Mean score: 0.917
-----
R2      :0.88
MAE      :94561.67
MSE      :14454707209.61
RMSE     :120227.73
```

4. Sử dụng Feed Forward Neural Network và Recurrent Neural Network để giải quyết bài toán

5. Áp dụng kỹ thuật tránh Overfitting trên các mô hình


1. Linear Regresion

- Sử dụng kỹ thuật regularization như L1 hoặc L2 regularization. Scikit-learn's Linear Regression có hỗ trợ cả hai. Bạn có thể thiết lập tham số alpha để kiểm soát mức độ regularization.

- Ridge Regression: là một phương pháp hồi quy tuyến tính với thêm vào hàm mất mát một thành phần regularization L2. Thành phần này giúp kiểm soát quá mức phức tạp của mô hình bằng cách thêm vào tổng của bình phương các trọng số (weights) vào hàm mất mát. Tham số alpha (còn được gọi là lambda) kiểm soát mức độ regularization, và giá trị alpha càng lớn thì độ "shrinkage" (coi nhẹ) của các trọng số càng mạnh.
- Lasso Regression: là một phương pháp hồi quy tuyến tính, nhưng thêm vào hàm mất mát một thành phần regularization L1. Thành phần này giúp giữ một số trọng số của đặc trưng chính xác bằng 0, từ đó có thể thực hiện lựa chọn đặc trưng tự động (feature selection). Tham số alpha kiểm soát mức độ regularization, và giá trị alpha càng lớn, regularization càng mạnh.

Code

- Tránh overfitting bằng L2:

```
 # Tránh overfitting bằng L2
# List to maintain the different cross-validation scores
cross_val_scores_ridge = []
# List to maintain the different values of alpha
alpha = []
# Loop to compute the different values of cross-validation scores
for i in range(1, 9):
    ridgeModel = Ridge(alpha = i * 0.25)
    ridgeModel.fit(X_train, y_train)
    scores = cross_val_score(ridgeModel, X, y, cv = 10)
    avg_cross_val_score = mean(scores)*100
    cross_val_scores_ridge.append(avg_cross_val_score)
    alpha.append(i * 0.25)
# Loop to print the different values of cross-validation scores
for i in range(0, len(alpha)):
    print(str(alpha[i])+' : '+str(cross_val_scores_ridge[i]))
# Building and fitting the Ridge Regression model
ridgeModelChosen = Ridge(alpha = 2)
ridgeModelChosen.fit(X_train, y_train)

# Evaluating the Ridge Regression model
print('Ridge Regression :',ridgeModelChosen.score(X_test, y_test))
```

- Tránh overfitting bằng L1:

```

▶ #Tránh overfitting bằng L1
# List to maintain the cross-validation scores
cross_val_scores_lasso = []

# List to maintain the different values of Lambda
Lambda = []

# Loop to compute the cross-validation scores
for i in range(1, 9):
    lassoModel = Lasso(alpha = i * 0.25, tol = 0.0925)
    lassoModel.fit(X_train, y_train)
    scores = cross_val_score(lassoModel, X, y, cv = 10)
    avg_cross_val_score = mean(scores)*100
    cross_val_scores_lasso.append(avg_cross_val_score)
    Lambda.append(i * 0.25)

# Loop to print the different values of cross-validation scores
for i in range(0, len(alpha)):
    print(str(alpha[i])+ ' : '+str(cross_val_scores_lasso[i]))

# Building and fitting the Lasso Regression Model
lassoModelChosen = Lasso(alpha = 2, tol = 0.0925)
lassoModelChosen.fit(X_train, y_train)

# Evaluating the Lasso Regression model
print('Lasso Regression model:',lassoModelChosen.score(X_test, y_test))

```

Kết quả chạy:

- Tránh overfitting bằng L2:

```

👤 0.25 : 91.73792634265821
    0.5 : 91.73792623714465
    0.75 : 91.73792589892555
    1.0 : 91.73792532809412
    1.25 : 91.73792452474349
    1.5 : 91.73792348896674
    1.75 : 91.73792222085694
    2.0 : 91.73792072050713
    Ridge Regression : 0.9179856303590432

```

- Tránh overfitting bằng L1:

```

0.25 : 91.73593195417729
0.5 : 91.73593287243243
0.75 : 91.7359337903172
1.0 : 91.73593470783162
1.25 : 91.73593562497567
1.5 : 91.73593654174937
1.75 : 91.73593745815269
2.0 : 91.73593837418568
Lasso Regression model: 0.9180758804902455

```

- Kết quả tổng quan:

```

[ ] # Building the two lists for visualization
models = ['Linear Regression', 'Ridge Regression', 'Lasso Regression']
scores = [model.score(X_test, y_test),
          ridgeModelChosen.score(X_test, y_test),
          lassoModelChosen.score(X_test, y_test)]

# Building the dictionary to compare the scores
mapping = {}
mapping['Linear Regression'] = model.score(X_test, y_test)
mapping['Ridge Regression'] = ridgeModelChosen.score(X_test, y_test)
mapping['Lasso Regression'] = lassoModelChosen.score(X_test, y_test)

# Printing the scores for different models
for key, val in mapping.items():
    print(str(key)+' : '+str(val))

Linear Regression : 0.917997170683436
Ridge Regression : 0.9179856303590432
Lasso Regression : 0.9180758804902455

```

2. DECISION TREES

Pruning (cắt tỉa) là một kỹ thuật regularization được sử dụng trong Decision Trees để tránh overfitting. Trong quá trình xây dựng Decision Trees, sau khi mọi điểm trong tập huấn luyện đều được phân lớp đúng, các leaf node có chung một non-leaf node sẽ được cắt tỉa. Non-leaf node đó sau cùng trở thành một leaf node mới, với class tương ứng là class chiếm đa số trong số các điểm dữ liệu thuộc về node đó.

Có hai phương pháp chính để thực hiện pruning:

- Pruning Dựa Trên Validation Set
- Pruning Dựa Trên Toàn Bộ Dataset

Trong đoạn code dưới đây, kỹ thuật pruning được thực hiện bằng cách sử dụng thư viện “scikit-learn” trong Python. Cụ thể, DecisionTreeRegressor từ “sklearn.tree” được sử dụng để xây dựng Decision Trees với kỹ thuật pruning.

Code

```
# Decision Trees with Pruning
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import train_test_split, GridSearchCV, cross_val_score
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error
import numpy as np
x = data.drop(['Price'], axis = 1)
y = data.Price
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Training with Pruning
pruned_model = DecisionTreeRegressor(max_depth=3, random_state=42)
pruned_model.fit(X_train, y_train)

# Prediction
y_pred_pruned = pruned_model.predict(X_test)
# Accuracy
accuracies_pruned = cross_val_score(estimator=pruned_model, X=X_train, y=y_train, cv=10)
print(accuracies_pruned)
print('Mean score with pruning: %0.3f' % accuracies_pruned.mean())
print('-----')

# Evaluation with Pruning
r2_pruned = r2_score(y_test, y_pred_pruned)
mae_pruned = mean_absolute_error(y_test, y_pred_pruned)
mse_pruned = mean_squared_error(y_test, y_pred_pruned)
rmse_pruned = np.sqrt(mse_pruned)

print(' R2 with pruning: %0.2f' % r2_pruned)
print(' MAE with pruning: %0.2f' % mae_pruned)
print(' MSE with pruning: %0.2f' % mse_pruned)
print(' RMSE with pruning: %0.2f' % rmse_pruned)
```

Kết quả đánh giá mô hình

```
[0.39919217 0.51896355 0.48887674 0.45384719 0.52883543 0.51184924
 0.53064406 0.50242712 0.5402032 0.53220078]
Mean score with pruning: 0.501
-----
R2 with pruning: 0.49
MAE with pruning: 198205.42
MSE with pruning: 63291998913.97
RMSE with pruning: 251579.01
```

3. KNN

k-fold Cross Validation là một phương pháp chuẩn bị tập validation để đánh giá độ chính xác thuật toán.

Tổng quan về phương pháp này như sau: Từ tập dữ liệu train, phương pháp này sẽ chia ra thành k phần(fold). Trong đó, việc huấn luyện chỉ được thực hiện trên k-1 fold, và chừa lại một test fold để thuật toán đó dự đoán thử. Và công việc này sẽ được thực hiện lặp lại nhiều lần để đảm bảo mỗi fold sẽ được trở thành test fold. Qua đó ta sẽ có k độ chính xác khác nhau, và kết quả ta thấy được chính là trung bình cộng của k độ chính xác đó.

Ta sử dụng thư viện “scikit-learn” với các modules như KNeighborsRegressor, cross_val_score, KFold, và các hàm đánh giá hiệu suất như r2_score, mean_absolute_error, mean_squared_error để thực hiện và đánh giá mô hình.

Code

```
#KNeighborsRegressor áp dụng k-fold cross-validation
from sklearn.neighbors import KNeighborsRegressor
from sklearn.model_selection import cross_val_score, KFold
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error
import numpy as np

# Giả sử 'X' và 'y' là các đặc trưng và biến mục tiêu của bạn
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Tạo mô hình KNeighborsRegressor
Model4 = KNeighborsRegressor(n_neighbors=1)

# Định nghĩa số lượng folds cho cross-validation
num_folds = 10

# Tạo đối tượng KFold
kf = KFold(n_splits=num_folds, shuffle=True, random_state=42)

# Thực hiện cross-validation
accuracies = cross_val_score(estimator=Model4, X=X_train, y=y_train, cv=kf)

# In các điểm số của cross-validation
print('Điểm số Cross-Validation:')
print(accuracies)
print('Điểm số trung bình: %0.3f' % accuracies.mean())
print('-----')

# Huấn luyện mô hình trên toàn bộ tập huấn luyện
Model4.fit(X_train, y_train)
```

```

# Dự đoán trên tập kiểm thử
y_pred4 = Model4.predict(X_test)

# Đánh giá mô hình
r2 = r2_score(y_test, y_pred4)
mae = mean_absolute_error(y_test, y_pred4)
mse = mean_squared_error(y_test, y_pred4)
rmse = np.sqrt(mse)

print('Đánh giá trên tập kiểm thử:')
print(' R2      : %.2f' % r2)
print(' MAE      : %.2f' % mae)
print(' MSE      : %.2f' % mse)
print(' RMSE     : %.2f' % rmse)

```

Kết quả đánh giá mô hình

```

Điểm số Cross-Validation:
[0.14425805 0.09815797 0.12422762 0.23827117 0.17066239 0.17683762
 0.21843261 0.22442577 0.09712149 0.17461026]
Điểm số trung bình: 0.167
-----
Đánh giá trên tập kiểm thử:
R2      : 0.17
MAE      : 252262.24
MSE      : 101590678265.35
RMSE     : 318732.93

```


REFERENCES

1. <https://analyticsindiamag.com/practical-approach-to-dimensionality-reduction-using-pca-lda-and-kernel-pca/>.
2. <https://www.cliffsnotes.com/study-guides/statistics/principles-of-testing/univariate-tests-an-overview>.
3. <https://www.blog.trainindata.com/recursive-feature-elimination-with-python/>.
4. <https://corporatefinanceinstitute.com/resources/data-science/lasso/>.
5. <https://www.educba.com/backward-elimination/>.
6. <https://www.statisticshowto.com/forward-selection/>.
7. <https://stats.stackexchange.com/questions/421361/how-exactly-does-bidirectional-stepwise-elimination-work>
8. <https://www.britannica.com/plant/tree/Popular-classifications>

