**Bài 3: Chuẩn bị dữ liệu - Lập trình AI bằng Python**

Chúng tôi đã nghiên cứu các thuật toán học máy có giám sát cũng như không được giám sát. Các thuật toán này yêu cầu dữ liệu được định dạng để bắt đầu quá trình đào tạo. Chúng ta phải chuẩn bị hoặc định dạng dữ liệu theo một cách nhất định để nó có thể được cung cấp làm đầu vào cho các thuật toán ML.

Bài này tập trung vào việc chuẩn bị dữ liệu cho các thuật toán học máy.

**1. Tiền xử lý dữ liệu :**

Trong cuộc sống hàng ngày, chúng ta xử lý rất nhiều dữ liệu nhưng dữ liệu này ở dạng thô. Để cung cấp dữ liệu làm đầu vào của các thuật toán học máy, chúng ta cần chuyển nó thành một dữ liệu có ý nghĩa. Đó là nơi mà quá trình tiền xử lý dữ liệu trở thành hình ảnh. Nói cách khác, chúng ta có thể nói rằng trước khi cung cấp dữ liệu cho các thuật toán học máy, chúng ta cần phải xử lý trước dữ liệu.

**Các bước xử lý dữ liệu :**

*Bước 1* - **Nhập các useful packages**- Nếu chúng ta đang sử dụng Python thì đây sẽ là bước đầu tiên để chuyển đổi dữ liệu sang một định dạng nhất định, tức là tiền xử lý. Nó có thể được thực hiện như sau:

**import** numpy as np

**import** sklearn.preprocessing

Ở đây chúng ta đã sử dụng 2 thư viện :

* **NumPy**- Về cơ bản NumPy là một gói xử lý mảng mục đích chung được thiết kế để thao tác hiệu quả các mảng đa chiều lớn của các bản ghi tùy ý mà không phải hy sinh quá nhiều tốc độ cho các mảng đa chiều nhỏ.
* Sklearn.preprocessing - thư viện này cung cấp nhiều chức năng tiện ích phổ biến và các lớp biến áp để thay đổi các vectơ đặc trưng thô thành một biểu diễn phù hợp hơn với các thuật toán học máy.

*Bước 2* - **Xác định dữ liệu mẫu** - Sau khi nhập các gói, chúng ta cần xác định một số dữ liệu mẫu để có thể áp dụng các kỹ thuật tiền xử lý trên dữ liệu đó. Bây giờ chúng ta sẽ xác định dữ liệu mẫu sau:

input\_data = np.array([2.1, -1.9, 5.5],

[-1.5, 2.4, 3.5],

[0.5, -7.9, 5.6],

[5.9, 2.3, -5.8])

*Bước 3*-**Áp dụng kỹ thuật tiền xử lý** - Trong bước này, chúng ta cần áp dụng bất kỳ kỹ thuật tiền xử lý nào.

Phần sau đây mô tả các kỹ thuật tiền xử lý dữ liệu.

**2. Kỹ thuật tiền xử lý dữ liệu :**

Các kỹ thuật để xử lý trước dữ liệu được mô tả dưới đây:

**a. Binarization :**

Đây là kỹ thuật tiền xử lý được sử dụng khi chúng ta cần chuyển đổi các giá trị số thành các giá trị Boolean. Chúng ta có thể sử dụng một phương thức có sẵn để mã hóa dữ liệu đầu vào bằng cách sử dụng 0,5 làm giá trị ngưỡng theo cách sau:

data\_binarized = preprocessing.Binarizer(threshold = 0.5).transform(input\_data)

print("\nBinarized data:\n", data\_binarized)

Bây giờ, sau khi chạy đoạn mã trên, chúng ta sẽ nhận được kết quả sau, tất cả các giá trị trên 0,5 (giá trị ngưỡng) sẽ được chuyển thành 1 và tất cả các giá trị dưới 0,5 sẽ được chuyển thành 0.

*Binarized data :*

[[ 1. 0. 1.]

[ 0. 1. 1.]

[ 0. 0. 1.]

[ 1. 1. 0.]]

**b, Mean Removal**

Đây là một kỹ thuật tiền xử lý rất phổ biến khác được sử dụng trong học máy. Về cơ bản, nó được sử dụng để loại bỏ giá trị trung bình khỏi vectơ đặc trưng để mọi đối tượng đều được căn giữa bằng 0. Chúng tôi cũng có thể xóa sai lệch khỏi các đối tượng trong vectơ đối tượng. Để áp dụng kỹ thuật tiền xử lý loại bỏ trung bình trên dữ liệu mẫu, chúng ta có thể viết mã Python được hiển thị bên dưới. Mã sẽ hiển thị độ lệch trung bình và độ lệch chuẩn của dữ liệu đầu vào -

print("Mean = ", input\_data.mean(axis = 0))

print("Std deviation = ", input\_data.std(axis = 0))

Đầu ra :

Mean = [ 1.75 -1.275 2.2]

Std deviation = [ 2.71431391 4.20022321 4.69414529]

Bây giờ, đoạn mã dưới đây sẽ loại bỏ Độ lệch trung bình và Độ lệch chuẩn của dữ liệu đầu vào  :

data\_scaled = preprocessing.scale(input\_data)

print("Mean =", data\_scaled.mean(axis=0))

print("Std deviation =", data\_scaled.std(axis = 0))

Đầu ra :

Mean = [ 1.11022302e-16 0.00000000e+00 0.00000000e+00]

Std deviation = [ 1. 1. 1.]

**c. Scaling :**

Nó là một kỹ thuật tiền xử lý dữ liệu khác được sử dụng để chia tỷ lệ các vectơ đặc trưng. Chia tỷ lệ của vectơ đối tượng là cần thiết vì giá trị của mọi đối tượng có thể khác nhau giữa nhiều giá trị ngẫu nhiên. Nói cách khác, chúng ta có thể nói rằng việc mở rộng quy mô là quan trọng bởi vì chúng tôi không muốn bất kỳ đối tượng địa lý nào là tổng hợp lớn hay nhỏ. Với sự trợ giúp của mã Python sau đây, chúng ta có thể thực hiện điều chỉnh tỷ lệ dữ liệu đầu vào của mình, tức là vector đặc trưng :

*#Min mã scaling :*

data\_scaler\_minmax = preprocessing.MinMaxScaler(feature\_range=(0,1))

data\_scaled\_minmax = data\_scaler\_minmax.fit\_transform(input\_data)

print ("\nMin max scaled data:\n", data\_scaled\_minmax)

**Min mã scaled data**

[ [ 0.48648649 0.58252427 0.99122807]

[ 0. 1. 0.81578947]

[ 0.27027027 0. 1. ]

[ 1. 0. 99029126 0. ]]

**d. Normalization :**

Nó là một kỹ thuật tiền xử lý dữ liệu khác được sử dụng để sửa đổi các vectơ đặc trưng. Loại sửa đổi như vậy là cần thiết để đo các vectơ đặc trưng trên một tỷ lệ chung. Tiếp theo là hai loại chuẩn hóa có thể được sử dụng trong học máy :

*L1 Normalization :*

Nó cũng được gọi là Sai lệch Tuyệt đối Ít nhất. Loại chuẩn hóa này sửa đổi các giá trị để tổng các giá trị tuyệt đối luôn lên đến 1 trong mỗi hàng. Nó có thể được triển khai trên dữ liệu đầu vào với sự trợ giúp của mã Python sau:

# Normalize data

data\_normalized\_l1 = preprocessing.normalize(input\_data, norm = 'l1')

print("\nL1 normalized data:\n", data\_normalized\_l1)

kết quả :

L1 normalized data:

[[ 0.22105263 -0.2 0.57894737]

[ -0.2027027 0.32432432 0.47297297]

[ 0.03571429 -0.56428571 0.4 ]

[ 0.42142857 0.16428571 -0.41428571]]

*L2 Normalization :*

Nó cũng được gọi là bình phương nhỏ nhất. Loại chuẩn hóa này sửa đổi các giá trị để tổng các ô vuông luôn lên đến 1 trong mỗi hàng. Nó có thể được triển khai trên dữ liệu đầu vào với sự trợ giúp của mã Python sau:

# Normalize data

data\_normalized\_l2 = preprocessing.normalize(input\_data, norm = 'l2')

print("\nL2 normalized data:\n", data\_normalized\_l2)

Kết quả :

L2 normalized data:

[[ 0.33946114 -0.30713151 0.88906489]

[ -0.33325106 0.53320169 0.7775858 ]

[ 0.05156558 -0.81473612 0.57753446]

[ 0.68706914 0.26784051 -0.6754239 ]]

**3. Label trong dữ liệu :**

Chúng ta đã biết rằng dữ liệu ở một định dạng nhất định là cần thiết cho các thuật toán học máy. Một yêu cầu quan trọng khác là dữ liệu phải được dán nhãn thích hợp trước khi gửi nó làm đầu vào của các thuật toán học máy. Ví dụ, nếu chúng ta nói về phân loại, có rất nhiều nhãn trên dữ liệu. Các nhãn đó ở dạng từ, số, v.v. Các chức năng liên quan đến học máy trong sklearn mong rằng dữ liệu phải có nhãn số. Do đó, nếu dữ liệu ở dạng khác thì nó phải được chuyển đổi thành số. Quá trình biến đổi các nhãn từ thành dạng số được gọi là mã hóa nhãn.

Các bước encodeing label:

*Bước 1* : import package :

**import** numpy as np

from sklearn **import** preprocessing

*Bước 2* : Định nghĩa label mẫu :

# Sample input labels

input\_labels = ['red','black','red','green','black','yellow','white']

Bước 3 : Khởi tạo và huấn luyện label :

Trong bước này, chúng ta cần tạo bộ mã hóa nhãn và đào tạo nó. Đoạn mã Python sau sẽ giúp thực hiện việc này:

# Creating the label encoder

**encoder** = preprocessing.LabelEncoder()

encoder.fit(input\_labels)

Kết quả :

**LabelEncoder**()

*Bước 4* : Kiểm tra hiệu suất bằng cách mã hóa danh sách có thứ tự ngẫu nhiên

Bước này có thể được sử dụng để kiểm tra hiệu suất bằng cách mã hóa danh sách được sắp xếp ngẫu nhiên:

# encoding a set of labels

test\_labels = ['green','red','black']

encoded\_values = encoder.transform(test\_labels)

print("\nLabels =", test\_labels)

Các nhãn sẽ được in như sau:

Labels = ['green', 'red', 'black']

Bây giờ, chúng ta có thể nhận được danh sách các giá trị được mã hóa, tức là các nhãn từ được chuyển đổi thành số như sau:

**print**("Encoded values =", **list**(encoded\_values))

Kết quả :

Encoded values = [1, 2, 0]

*Bước 5*- Kiểm tra hiệu suất bằng cách giải mã một bộ số ngẫu nhiên -

# decoding a set of values

encoded\_values = [3,0,4,1]

decoded\_list = encoder.inverse\_transform(encoded\_values)

print("\nEncoded values =", encoded\_values)

Bây giờ, các giá trị được mã hóa sẽ được in như sau:

Encoded values = [3, 0, 4, 1]

print("\nDecoded labels =", list(decoded\_list))

Các giá trị được giải mã sẽ được in:

**Decoded** labels = ['white', 'black', 'yellow', 'green']

**4. Dữ liệu gắn nhãn và không gắn nhãn :**

Dữ liệu không được gắn nhãn chủ yếu bao gồm các mẫu vật thể tự nhiên hoặc do con người tạo ra có thể dễ dàng lấy được từ thế giới. Chúng bao gồm, âm thanh, video, ảnh, tin bài, v.v.

Mặt khác, dữ liệu được gắn nhãn lấy một tập hợp dữ liệu không được gắn nhãn và tăng cường mỗi phần của dữ liệu không được gắn nhãn đó bằng một số thẻ hoặc nhãn hoặc lớp có ý nghĩa. Ví dụ: nếu chúng ta có một bức ảnh thì nhãn có thể được đặt dựa trên nội dung của bức ảnh, tức là ảnh của một cậu bé hoặc cô gái hoặc động vật hoặc bất kỳ thứ gì khác. Việc gắn nhãn dữ liệu cần có chuyên môn hoặc sự đánh giá của con người về một phần dữ liệu chưa được gắn nhãn nhất định.

Có nhiều tình huống trong đó dữ liệu không được gắn nhãn rất phong phú và dễ dàng lấy được nhưng dữ liệu được gắn nhãn thường yêu cầu con người / chuyên gia chú thích. Học tập bán giám sát cố gắng kết hợp dữ liệu được gắn nhãn và không được gắn nhãn để xây dựng các mô hình tốt hơn.