ĐẠI HỌC QUỐC GIA TP.HCM

TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN

CS116. Python cho Machine Learning

KHOA: KHOA HỌC MÁY TÍNH

LT01 – CHUẨN HÓA GIÁ TRỊ LIÊN TỤC

Đề tài: Có những phương pháp chuẩn hóa giá trị liên tục nào trong sklearn? Công thức từng phương pháp chuẩn hóa là gì? Khi nào ta nên sử dụng hàm chuẩn hóa nào?

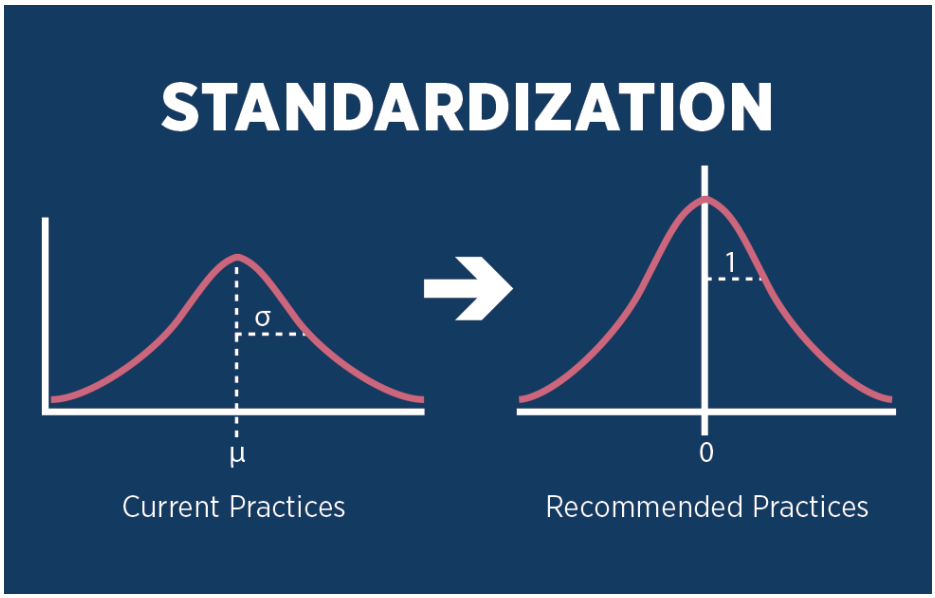
GV hướng dẫn: TS. Nguyễn Vinh Tiệp

Thành viên thực hiện:

1. Huỳnh Viết Tuấn Kiệt – 20521494

01.01 Tại sao phải chuẩn hóa dữ liệu?

– Chuẩn hóa dữ liệu ***(data normalization)*** là một kĩ thuật chuẩn bị dữ liệu phổ biến trong Machine Learning. Mục tiêu của chuẩn hóa dữ liệu là biến đổi các đặc trưng về cùng một tỉ lệ ***(scales)*** hay phân phối ***(distributions)***. Nói cách khác, phương pháp này làm thay đổi phạm vi của các giá trị ***(change the range of the values)*** giúp cải thiện hiệu suất và độ ổn định huấn luyện của một mô hình.



Hình minh họa: Chuẩn hóa được sử dụng để biến đổi dữ liệu về cùng một phân phối.

01.02 Các phương pháp chuẩn hóa giá trị liên tục trong sklearn – Công thức – Trường hợp sử dụng:

01.02.01 Min – max Normalization

- **Min – max** là một kỹ thuật chia tỷ lệ ***(scaling)*** trong đó các giá trị được thay đổi tỷ lệ và dịch chuyển để chúng nằm trong khoảng từ 0 đến 1.
- Công thức:

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)}$$

  + Trong đó:
    - x là giá trị ban đầu chưa được chuẩn hóa
    - x' là giá trị chuẩn hóa
    - min là giá trị nhỏ nhất trong x
    - max là giá trị lớn nhất trong x

- Ví dụ:

```
[1]  1  from sklearn.preprocessing import MinMaxScaler

[2]  1  data = [[3, 2], [15, 6], [0, 10], [1, 18]]

[3]  1  scaler = MinMaxScaler()
     2  print(scaler.fit(data))

     MinMaxScaler()

[4]  1  print(scaler.data_max_)

     [15. 18.]

[5]  1  print(scaler.data_min_)

     [0. 2.]

[6]  1  print(scaler.transform(data))

     [[0.2        0.        ]
      [1.         0.25      ]
      [0.         0.5       ]
      [0.06666667 1.        ]]
```

- Trường hợp sử dụng:
  + **Min – max normalization** được ưu tiên sử dụng khi dữ liệu không tuân theo phân phối Gaussian hoặc phân phối chuẩn. Nó được ưu tiên cho việc chuẩn hóa các thuật toán không tuân theo bất kỳ phân phối nào, chẳng hạn như KNN và Neural Network.

01.02.02 Z-score Normalization

- Chuẩn hóa **z-score** hay còn gọi là **Standardization** xem các đặc trưng được thay đổi tỉ lệ theo cách tuân theo thuộc tính phân phối chuẩn với $\mu = 0$ và $\sigma = 1$. Trong đó μ là trung bình (average) và σ là độ lệch chuẩn so với trung bình.
- Công thức

$$x' = \frac{x - \mu}{\sigma}$$

  + Trong đó:
    - x là giá trị ban đầu chưa được chuẩn hóa
    - x' là giá trị chuẩn hóa
    - μ là trung bình của các giá trị
    - σ là độ lệch chuẩn của các giá trị

- Ví dụ:

```
[7]  1  from sklearn.preprocessing import StandardScaler

[8]  1  data = [[3, 2], [15, 6], [0, 10], [1, 18]]

[12] 1  scaler = StandardScaler()
     2  print(scaler.fit(data))

     StandardScaler()

[13] 1  print(scaler.mean_)

     [4.75 9.  ]

[14] 1  print(scaler.transform(data))

     [[-0.29091007 -1.18321596]
      [ 1.70390185 -0.50709255]
      [-0.78961305  0.16903085]
      [-0.62337873  1.52127766]]
```

- Trường hợp sử dụng:
  + **Z-score** có thể hữu ích trong các trường hợp dữ liệu tuân theo phân phối Gaussian hoặc thậm chí không. Standardization không có phạm vi giới hạn, nghĩa là ngay cả khi có những giá trị ngoại lệ trong dữ liệu, chúng sẽ không bị ảnh hưởng bởi quá trình **standardization**

01.02.03 Log scaling

- **Log scaling** tính toán log của các giá trị để nén một phạm vi rộng thành một phạm vi hẹp. Nói cách khác, nó giúp chuyển đổi phân phối lệch thành phân phối chuẩn hoặc phân phối ít lệch hơn. Để thực hiện chia tỷ lệ, lấy log các giá trị và biến thay thế chúng thành giá trị mới.
- Công thức:

$$x' = \log(x)$$

  + Trong đó:
    - x là giá trị ban đầu chưa được chuẩn hóa
    - x' là giá trị chuẩn hóa
- Ví dụ:

```
[15]  1  import numpy as np
      2  from sklearn.preprocessing import FunctionTransformer

[16]  1  data = np.array([[0,1], [2,3]])

[17]  1  transformer = FunctionTransformer(np.log1p)

[18]  1  print(transformer.transform(data))

[[0.         0.69314718]
 [1.09861229 1.38629436]]
```

- Trường hợp sử dụng
  + **Log scaling** phù hợp với bộ dữ liệu chứa giá trị ngoại lệ (giá trị khác biệt so với phần còn lại) lớn.

01.02.04 Maximum Absolute Scaling

- **MaxAbsScaler** tương đồng với **MinMaxScaler** nhưng phương pháp này chia giá trị trong khoảng $[-1,1]$ bằng cách chia cho giá trị lớn nhất trong mỗi đặc trưng.
- Công thức:

$$x' = \frac{x}{\max(|x|)}$$

  + Trong đó:
    - x là giá trị ban đầu chưa được chuẩn hóa
    - x' là giá trị chuẩn hóa
- Ví dụ:

```
[19]  1  from sklearn.preprocessing import MaxAbsScaler

[24]  1  data = [[ 1., -1.,  2.],
      2          [ 2.,  0.,  0.],
      3          [ 0.,  1., -1.]]

[25]  1  scaler = MaxAbsScaler()
      2  print(scaler.fit(data))

     MaxAbsScaler()

[26]  1  print(scaler.max_abs_)

     [2. 1. 2.]

[27]  1  print(scaler.transform(data))

     [[ 0.5 -1.   1. ]
      [ 1.   0.   0. ]
      [ 0.   1.  -0.5]]
```

- Trường hợp sử dụng:
  + **MaxAbsScaler** có ý nghĩa khi áp dụng lên dữ liệu đã được căn giữa ở mức 0 hoặc dữ liệu thưa thớt ***(sparse data)***

01.02.05 RobustScaler

- **RobustScaler** hoạt động sử dụng trung vị và phần tư, điều này làm cho bộ chia tỷ lệ bỏ qua các điểm dữ liệu ngoại lệ (rất khác biệt) so với phần còn lại
- Công thức:

$$x' = \frac{x - x_{med}}{x_{75} - x_{25}}$$

  + Trong đó:
    - x là giá trị ban đầu chưa được chuẩn hóa
    - x' là giá trị chuẩn hóa
    - x_{med} là giá trị trung vị của x
    - x_{75} là giá trị liên phần 75%
    - x_{25} là giá trị liên phần 25%
- Ví dụ:

```
1s  1  from sklearn.preprocessing import RobustScaler

[29]  1  data = [[ 1., -2.,  2.],
      2          [ -2.,  1.,  3.],
      3          [ 4.,  1., -2.]]

[30]  1  scaler = RobustScaler()
      2  print(scaler.fit(data))

      RobustScaler()

[33]  1  print(scaler.center_)

      [1. 1. 2.]

[34]  1  print(scaler.transform(data))

      [[ 0.  -2.   0. ]
       [-1.   0.   0.4]
       [ 1.   0.  -1.6]]
```

- Trường hợp sử dụng:
  + **RobustScaler** phù hợp cho dữ liệu có chứa các điểm dữ liệu ngoại lệ và muốn giảm ảnh hưởng của các điểm dữ liệu đó trong quá trình huấn luyện.

01.02.06 Normalizer

- **Normalizer** được sử dụng để chia tỷ lệ tập dữ liệu đầu vào trên thang từ 0 đến 1 để có định mức đơn vị (unit norm). Norm không là gì khác ngoài việc tính toán độ lớn của vecto.
- Công thức:

$$x' = \frac{x}{\|x_n\|_i}$$

  + Nếu $norm = l1 \rightarrow \|x_n\|_1 = |x_1| + |x_2| + \cdots + |x_n|$
  + Nếu $norm = l2 \rightarrow \|x_n\|_2 = \sqrt{x_1^2 + x_2^2 + \cdots + x_n^2}$
  + Nếu $norm = max \rightarrow \|x_n\|_\infty = \max\{|x_1|, |x_2|, \ldots, |x_n|\}$
- Ví dụ:

```
[35]  1  from sklearn.preprocessing import Normalizer

[43]  1  data = [[4, 1, 2, 2],
      2           [1, 3, 9, 3],
      3           [5, 7, 5, 1]]

      1  trans_l1 = Normalizer(norm='l1')
      2  trans_l2 = Normalizer()
      3  trans_max = Normalizer(norm='max')
      4  print(trans_l1.fit(data))
      5  print(trans_l2.fit(data))
      6  print(trans_max.fit(data))

Normalizer(norm='l1')
Normalizer()
Normalizer(norm='max')

[49]  1  print(trans_l1.transform(data))

[[0.44444444 0.11111111 0.22222222 0.22222222]
 [0.0625     0.1875     0.5625     0.1875    ]
 [0.27777778 0.38888889 0.27777778 0.05555556]]

[50]  1  print(trans_l2.transform(data))

[[0.8 0.2 0.4 0.4]
 [0.1 0.3 0.9 0.3]
 [0.5 0.7 0.5 0.1]]

[51]  1  print(trans_max.transform(data))

[[1.         0.25       0.5        0.5       ]
 [0.11111111 0.33333333 1.         0.33333333]
 [0.71428571 1.         0.71428571 0.14285714]]
```

- Trường hợp sử dụng:
  + **Normalizer** hữu ích để kiểm soát kích thước của một vecto trong quá trình lặp đi lặp lại để tránh sự không ổn định do các giá trị số quá lớn.
  + Có ích hơn trong hồi quy **(regression)** so với phân loại **(classification)**