Bài tập thực hành-Khai thác dữ liệu-tuần 2

Phan Hồng Trâm - 21110414

April 2024

Mục lục

1 Viết hàm tính các chuẩn $p = 1, 2, \infty$ cho 50 dòng đầu tiên của array trong mục 1.

```
from numpy.linalg import norm
def calculate_Lp_norm(array, p):
  for i in range(0,50):
    for j in range(1,50):
      dist = norm(array[i,:] - array[j,:], p)
      print(f"Dist({i+1},{j+1}) with p = {p} is: ", dist)
  return dist
```

Giải thích code:

- Ta import thư viện `from numpy.linalg import norm` để tính khoảng cách theo chuẩn `Lp`.
- Hàm `def calculate_Lp_norm` nhận 2 tham số đầu vào là `array` và `p`, với:
  - `array`: Mảng numpy cần tính toán.
  - `p`: Giá trị của p $(1, 2, \infty)$.
- Hàm lặp qua 50 dòng đầu tiên của mảng `array` và tính toán chuẩn `p` cho từng cặp dòng.
- Hàm `norm` giúp tính khoảng cách giữa 2 điểm dữ liệu.
- Sau đó dùng hàm `print` để in ra khoảng cách của từng cặp điểm dữ liệu.

Sau đó ta gọi hàm với `p = 1`, và in ra kết quả (xem trong file code). Tương tự với `p = 2` và `p = inf`.

```
# P = 1
L1_norm = calculate_Lp_norm(array, 1)
```

```
# p = 2
L2_norm = calculate_Lp_norm(array, 2)
```

```
# p = inf
Linf_norm = calculate_Lp_norm(array, np.inf)
```

2 Tính các láng giềng gần nhất cho 100 dòng đầu tiên của array ở mục 2 sử dụng mục 2 với độ đo tương đồng và độ đo tần suất xuất hiện ngược.

Tương tự như bài 1, ta vẫn đọc file dữ liệu, cleaning data (check null, drop object columns, ...), chuẩn hóa dữ liệu bằng `StandardScaler`, rời rạc hóa dữ liệu bằng hàm `KBinsDiscretizer` (xem ở file code), sau đó tiến hành tính các láng giềng gần nhất cho 100 dòng đầu tiên.

2.1 a. Sử dụng độ đo tương đồng

Lý thuyết:
Xét 2 bản ghi $\overline{X} = (x_1 \dots x_d)$ và $\overline{Y} = (y_1 \dots y_d)$, sự tương đồng đơn giản nhất giữa hai bản ghi này được xác định như sau:

$$Sim(\overline{X}, \overline{Y}) = \sum_{i=1}^{d} S(x_i, y_i). \tag{1}$$

Với: $S(x_i, y_i)$ là sự tương đồng giữa các giá trị thuộc tính x_i, y_i Lựa chọn đơn giản nhất cho $S(x_i, y_i)$ là:

$$\begin{cases} S(x_i, y_i) = 1, \text{ if } x_i = y_i \\ S(x_i, y_i) = 0, \text{ in contrast} \end{cases}$$

Code:

```
def simple_matching_similarity(x, y):
  # Check if data records have the same length
  if np.array_equal(x, y):
    similarity = 1/len(x)
  else:
    similarity = 0
  return similarity

# Save the nearest neighbor variable of each data row
nearest_neighbors = []

# Loop through the first 100 data rows
for i in range (0,101):
  current_row = array[i]
  # Initialize the maximum similarity value and nearest neighbor
  max_simi = -1
  nearest_neighbor = None
  # Loop through other data points and calculate similarity
  for j in range(100):
    if i != j:
      similarity = simple_matching_similarity(current_row, array[j])
      if similarity > max_simi:
        max_simi = similarity
        nearest_neighbor = j
   # Save the nearest neighbor value for the current row
  nearest_neighbors.append(nearest_neighbor)

for i, neighbor in enumerate(nearest_neighbors):
  print(f"Nearest neighbor of row {i + 1} is row {neighbor + 1}")
```

Giải thích code:

- Định nghĩa hàm `simple_matching_similarity` nhận hai mảng dữ liệu `x` và `y` làm đầu vào và trả về độ tương đồng giữa hai bản ghi dữ liệu.
  - `np.array_equal` để kiểm tra xem hai mảng có bằng nhau hay không. Nếu có, độ tương đồng sẽ được đặt là 1/chiều dài của mảng. Nếu hai mảng không bằng nhau, độ tương đồng sẽ được đặt là 0.
- Khởi tạo list rỗng `nearest_neighbors` để chứa các láng giềng gần nhất của mỗi dòng.
- Vòng lặp `for i in range(0, 101)`: Vòng lặp này duyệt qua 100 hàng dữ liệu đầu tiên. Trong mỗi lần lặp, biến `current_row` sẽ lưu trữ hàng dữ liệu hiện tại.
- Khởi tạo biến `max_simi` và `nearest_neighbor`:
  - Biến `max_simi` được sử dụng để lưu trữ giá trị độ tương đồng lớn nhất được tìm thấy.
  - Biến `nearest_neighbor` được sử dụng để lưu trữ chỉ mục của hàng dữ liệu gần nhất với hàng dữ liệu hiện tại.
- Vòng lặp `for j in range(100)`: Vòng lặp này duyệt qua tất cả các hàng dữ liệu khác. Trong mỗi lần lặp, biến `j` sẽ lưu trữ chỉ mục của hàng dữ liệu hiện tại.
- Điều kiện `if i != j`: Điều kiện này đảm bảo rằng chúng ta không tính toán độ tương đồng giữa hàng dữ liệu hiện tại và chính nó.
- Ta tính toán độ tương đồng giữa hàng dữ liệu hiện tại và hàng dữ liệu tại chỉ mục `j` bằng cách gọi hàm `simple_matching_similarity` và lưu trữ trong biến `similarity`.
- Cập nhật lại biến `max_simi` và `nearest_neighbor`
- Lưu trữ giá trị `nearest_neighbor` trong danh sách `nearest_neighbors`
- Vòng lặp `for i, neighbor in enumerate(nearest_neighbors)`: duyệt qua danh sách `nearest_neighbors` và in ra chỉ mục của hàng dữ liệu gần nhất cho mỗi hàng dữ liệu.

Kết quả code: Xem trong file code.

2.2 b. Sử dụng độ đo tần suất xuất hiện ngược

Lý thuyết:
Cho $p_k(x)$ là một tỉ số của các bản ghi mà thuộc tính thứ k lấy giá trị x trong tập dữ liệu:

$$\begin{cases} S(x_i, y_i) = \dfrac{1}{p_k(x)^2}, \text{if } x_i = y_i \\ S(x_i, y_i) = 0, \text{in contrast} \end{cases}$$

Code:

```
def IFAM(x,y,k):
  # Calculate the frequency of occurrence of each value x[k] in the array x
  p_k_x = np.square(np.sum(x == x[k]))/len(x)
  # calculate the similarity
  if np.array_equal(x,y):
    similarity = 1 /(p_k_x**2)
  else:
    similarity = 0
  return similarity

# Initialize an empty list nearest_neighbors to store the nearest neighbors of each data row
nearest_neighbors = []

#  Loop through the first 100 data rows
for i in range (0,101):
  current_row = array[i]
  # Initialize the maximum similarity value and nearest neighbor
  max_simi = float('-inf')
  nearest_neighbor = None
  # Loop through other data points and calculate similarity
  for j in range(100):
    if i != j:
      for k in range(len(current_row)):
        similarity = IFAM(current_row, array[j],k)
        if similarity >= max_simi:
          max_simi = similarity
          nearest_neighbor = j
   # Save the nearest neighbor value for the current row
  nearest_neighbors.append(nearest_neighbor)

for i, neighbor in enumerate(nearest_neighbors):
  print(f"Nearest neighbor of row {i + 1} is row {neighbor + 1}")
```

Giải thích code:

- Hàm `IFAM` nhận ba tham số `x`, `y`, và `k`. Hàm này thực hiện các bước sau:
  - Tính toán tần suất xuất hiện của mỗi giá trị `x[k]` trong mảng `x` và lưu vào biến `p_k_x`. hàm `np.sum` để tính tổng số lần giá trị `x[k]` xuất hiện trong mảng `x`. Sau đó, nó sử dụng hàm `np.square` để bình phương giá trị này và chia cho độ dài của mảng `x`để tính toán tần suất xuất hiện.
  - Tính toán độ tương đồng: `np.array_equal` để kiểm tra xem hai mảng có bằng nhau hay không. Nếu có, độ tương đồng sẽ được đặt là `1 /(p_k_x**2`. Nếu hai mảng không bằng nhau, độ tương đồng sẽ được đặt là 0.
- Khởi tạo một danh sách rỗng lưu trữ các lân cận gần nhất cho mỗi dòng dữ liệu: `nearest_neighbors`
- Lặp qua 100 dòng dữ liệu đầu tiên: Trong mỗi lần lặp, biến `current_row` sẽ lưu trữ hàng dữ liệu hiện tại.

- Khởi tạo biến `max_simi` và `nearest_neighbor`: Dòng code này khởi tạo giá trị độ tương đồng tối đa `max_simi` bằng giá trị âm vô cùng và lân cận gần nhất `nearest_neighbor` bằng giá trị `None`.
- Vòng lặp `for j in range(100):` Vòng lặp này duyệt qua tất cả các hàng dữ liệu khác. Trong mỗi lần lặp, biến `j` sẽ lưu trữ chỉ mục của hàng dữ liệu hiện tại.
- Điều kiện `if i != j`: Điều kiện này đảm bảo rằng chúng ta không tính toán độ tương đồng giữa hàng dữ liệu hiện tại và chính nó.
- Vòng lặp này lặp qua tất cả các dòng dữ liệu khác trong tập dữ liệu. Đối với mỗi dòng dữ liệu `array[j]`, nó tính toán độ tương đồng giữa `current_row` và `array[j]` bằng cách sử dụng hàm `IFAM`. Nếu độ tương đồng này cao hơn hoặc bằng giá trị độ tương đồng tối đa hiện tại, nó cập nhật giá trị độ tương đồng tối đa và lân cận gần nhất.
- Lưu lân cận gần nhất cho dòng dữ liệu hiện tại bằng cách dùng hàm `append`.
- Vòng lặp `for i, neighbor in enumerate(nearest_neighbors):` duyệt qua danh sách `nearest_neighbors` và in ra chỉ mục của hàng dữ liệu gần nhất cho mỗi hàng dữ liệu.

Kết quả code: Xem trong file code.