초심자를 위한 생물학+정보학

_37_

- 다양한 거리 계산 -

고 주 온 (John Go, Ph.D.)

지난 번에 코사인 유사도를 확인하는 데 사용했던 파이썬의 `scipy` 모듈에서 거리 (distance)의 계산에 적용할 수 있는 다양한 함수들을 보면, 다음과 같다.

표 1. scipy 모듈의 거리 계산 함수

모듈명	내용 (특기하지 않는 한 2 개의 1차원 배열을 대상으로 함)
`braycurtis(u, v)`	브레이-커티스 거리 계산
`canberra(u, v)`	캔버라 거리 계산
`chebyshev(u, v)`	체비쇼프 (Chebyshev) 거리 (체스판 거리) 계산
`cityblock(u, v)`	맨해튼 (Manhattan) 거리 계산
`correlation(u, v)`	연관 거리 (correlation distance) 계산
`cosine(u, v)`	코사인 거리 (cosine distance) 계산
`dice(u, v)`	1차원 불 배열 (boolean 1-D arrays)*에 대한 다이스 비유사도 (dice dissimilarity) 계산
`euclidean(u, v)`†	유클리드 거리 (Euclidean distance) 계산
`hamming(u, v)`	해밍 거리 (hamming distance) 계산
`jaccard(u, v)`	1차원 불 배열에 대한 자카드-니덤 비유사도 (Jaccard-Needham dissimilarity) 계산
`kulsinski(u, v)`	1차원 불 배열에 대한 쿨친스키 비유사도 (Kulsinski dissimilarity) 계산
`mahalanobis(u, v, VI)`	마할라노비스 거리 (Mahalanobis distance) 계산
`matching(u, v)`	1차원 불 배열에 대한 매칭 비유사도 (matching dissimilarity) 계산
`minkowski(u, v, p)`	민코브스키 거리 (Minkowski distance) 계산
`rogerstanimoto(u, v)`	1차원 불 배열에 대한 로저스-타니모토 비유사도 (Rogers-Tanimoto dissimilarity) 계산
`russellrao(u, v)`	1차원 불 배열에 대한 러셀-라오 비유사도 (Russell-Rao dissimilarity) 계산
`seuclidean(u, v, V)`	표준 유클리드 거리 (standardized Euclidean distance) 계산
`sokalmichener(u, v)`	1차원 불 배열에 대한 소칼-미쉬너 비유사도 (Sokal-Michener dissimilarity) 계산
`sokalsneath(u, v)`	1차원 불 배열에 대한 소칼-스니스 비유사도 (Sokal-Sneath dissim-

	ilarity) 계산
sqeuclidean(u, v)	제곱 유클리드 거리 (squared Euclidean distance) 계산
wminkowski(u, v, p, w)	가중 민코브스키 거리 (weighted Minkowski distance) 계산
yule(u, v)	1차원 불 배열에 대한 율 비유사도 (Yule dissimilarity) 계산

* 0 또는 1만으로 이루어진 1차원 배열.

† 측정법 (metric)이 주어지지 않을 경우 기본값.

파이썬의 scipy 모듈에 있는 거리 계산 관련 함수는 거리 (distance)에 상응하는 개념인 비유사도 (非類似度, dissimilarity)에 관한 함수도 포함하는데, 표 1에서 볼 수 있는 바와 같이 scipy 모듈에서 사용할 수 있는 관련 함수는 20여 가지가 있다. 이들 거리, 비유사도 등의 개념은 개체의 분류 (classification) 또는 군집화 (clustering) 등의 수행에 필수적으로 적용되는 중요한 개념이다. 그런데, 여기서 짚고 넘어가야 할 중요한 사항이 있다. 자료 또는 각 객체 간의 관계를 확인하기 위해서는 보통 거리, 유사도, 상관 계수 (correlation coefficient) 등의 개념을 도입하여 이를 객관화하게 되는데, 이 과정에서 용어나 개념의 혼란이 있을 수 있다. 앞에서 본 코사인 유사도 (cosine similarity)와 코사인 거리 (cosine distance)를 예로 하여 scipy 모듈의 사용법과 유의 사항에 대해 알아 보자.

```
------
>>> A = [1, 1, 0, 1]  <-- (1)
>>> B = [2, 0, 1, 1]  <-- (2)
>>> import scipy.spatial.distance  <-- (3)
>>> scipy.spatial.distance.pdist((A, B), 'cosine')  <-- (4)
array([ 0.29289322])  <-- (5)
>>> pdist((A, B), 'cosine')  <-- (6)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'pdist' is not defined  <-- (7)
>>>
------
```

지난 번 코사인 유사도의 예에서 사용했던 자료를 재적용하고 (1, 2), scipy 모듈의 거리 계산 기능을 불러 온다 (3). pdist()를 사용하여 (4) 코사인 거리를 계산한다 (5). 모듈을 제외하고 pdist()만 실행할 경우에는 (6) 정의되지 않았다고 오류를 출력한다 (7). pdist()는 scipy 모듈이 포함한 함수를 실행할 때 n-차원 공간의 객체 간의 짝 거리 (pairwise distance)를 구하는 기능으로서, 예시한 바와 같이 1차원 배열 쌍과 적용하고자 하는 측정법을 인수 (argument)로 한다.

그런데, 이런 방식으로 실행을 하려면 기능을 호출할 때마다 매번 원래의 모듈명을 모두 입력해야 하는 번거로움이 있다. 그래서, 반복적인 계산을 간편하게 하기 위해 실제로 실행할 때에는 다음과 같은 몇 가지 방법을 사용하기도 한다.

```
------
>>> A = [1, 1, 0, 1]
>>> B = [2, 0, 1, 1]
>>> from scipy.spatial.distance import *  <-- (1)
>>> pdist((A, B), 'cosine')  <-- (2)
array([ 0.29289322])
>>>
```

```
------
```

scipy.spatial.distance 모듈로부터 (from) 모든 함수를 불러 온 다음 (import *, 1), pdist()를 실행하여 코사인 거리를 계산한다 (2). 위의 방법은 거리 모듈의 전체 함수를 불러 오는 (import *) 것이다. 또 다른 방법으로는 해당 모듈에 임의의 모듈명 (dist1, 다른 명칭도 상관없다)을 지정하여 다음과 같이 사용할 수도 있다.

```
------
>>> A = [1, 1, 0, 1]
>>> B = [2, 0, 1, 1]
>>> import scipy.spatial.distance as dist1  <-- (1)
>>> dist1.pdist((A, B), 'cosine')  <-- (2)
array([ 0.29289322])
>>>
------
```

scipy.spatial.distance 모듈을 dist1이라는 이름으로 불러온 후 (1), dist1에 대한 pdist()로 코사인 거리를 계산한다 (2). 참고로, pdist()를 사용하는 방법으로는 다음과 같은 것도 있다.

```
------
>>> A = [1, 1, 0, 1]
>>> B = [2, 0, 1, 1]
>>> from scipy.spatial.distance import pdist  <-- (1)
>>> pdist((A, B), 'cosine')  <-- (2)
array([ 0.29289322])
>>>
------
```

scipy.spatial.distance 모듈로부터 pdist함수를 불러 온 다음 (1), 이를 사용하여 A와 B의 코사인 거리를 계산한다 (2).

지금까지 모듈의 기본적인 호출 방법에 대해 약간 알아 보았는데, 여기서 기억해 두어야 할 내용은, 사용하고자 하는 함수를 포함하는 모듈명 전체를 호출하여 해당 함수를 사용해야 한다는 것이다. 모듈을 사용하는 초기에는 모듈을 불러오는 방법이 쉽게 익숙해지지 않아서 혼란스러울 수 있지만, 내친 김에 pdist()를 사용하지 않고 직접 함수를 실행하는 방법도 알아 보자. 아래 예시한 것들은 내용이 조금씩 차이가 나므로 주의깊게 보아야 한다.

```
------
>>> import scipy.spatial.distance  <-- (1)
>>> distance.cosine(A, B)  <-- (2)
0.29289321881345243  <-- (3)
>>>
------
```

scipy.spatial.distance 모듈을 호출하고 (1) 코사인 거리 계산 함수 (cosine(u, v))를 실행하면 (2), 결과가 실수 값 (부동 소수점 수)으로 출력된다 (3).

```
------
>>> from scipy.spatial import distance  <-- (1)
>>> distance.cosine(A, B)  <-- (2)
0.29289321881345243  <-- (3)
```

```
>>>
------
```

코사인 거리 함수를 포함하는 모듈의 호출 방식을 달리하여 (1) 코사인 거리를 계산해도 (2) 동일한 결과를 볼 수 있다 (3).

```
------
>>> import scipy.spatial.distance as dist2  <-- (1)
>>> dist2.cosine(A, B)  <-- (2)
0.29289321881345243  <-- (3)
>>>
------
```

불러 올 모듈을 임의의 이름으로 호출하여 (as dist2, 1) 코사인 거리를 계산하는 것도 가능하다 (2, 3).

```
------
>>> from scipy.spatial import distance as dist3  <-- (1)
>>> dist3.cosine(A, B)  <-- (2)
0.29289321881345243  <-- (3)
>>>
------
```

모듈 명칭의 지정을 달리하여도 (1) 동일한 결과를 보인다 (2, 3).

이처럼 다양한 방식으로 모듈을 불러 거리 계산을 실행할 수 있는데, 이는 파이썬 모듈 운영에 있어서 그 계층적 구조에 대해 잘 이해하면 쉽게 응용할 수 있다. 또한, 모듈 사용 방식에 대한 사용자의 취향도 중요하므로, 가능한 한 많은 경험을 쌓으면서 기존의 개발자들이 적용한 개발 표준을 참조하여 작업하는 것이 좋을 것이다.

지금까지 scipy 모듈을 이용한 코사인 거리 계산의 예를 통하여 사용 방법에 대해 간략히 알아 보았다. 그런데, 코사인 거리의 계산 결과를 보면, 지난 번의 코사인 유사도 (S_c) 계산 결과 (0.70710678)와 이번 코사인 거리 (D_c)의 결과 (0.29289322) 사이에 지난 번에 언급한 것과 같이 $D_c = 1 - S_c$의 관계가 성립되는 것을 볼 수 있다. 이를 달리 말하자면, 함수나 매개 변수의 명칭이 같은 cosine이라고 해도 주어진 기능에서의 개념이 무엇인지 반드시 확인해야 한다. 유사도 (similarity), 거리 (distance), 비유사도 (부동성, dissimilarity), 계수 (coefficient) 등의 표현도 해당 용어의 실제 개념과 의미에 따라서 다르게 사용될 수 있다. 파이썬의 scipy 모듈인 자카드-니덤 비유사도 (jaccard(u, v))와 로저스-타니모토 비유사도 (rogerstanimoto(u, v))는 일반적으로 언급되는 자카드 계수 (Jaccard coefficient)나 타니모토 계수 (Tanimoto coefficient)와는 각각 다른 개념이다. 내용이 다소 혼란스러울 수 있으나, 간단히 요약하자면 용어 및 그 개념에 대한 정의를 정확하게 확인하고 해당 기능을 유의해서 사용해야 오류를 피할 수 있다는 것이다.

참고로, scipy.spatial.distance 모듈을 사용한 거리 계산 시, 적용하고자 하는 측정법을 특정하지 않으면, 다음 예에서 보는 바와 같이, 유클리드 거리 계산을 기본값 (default)으로 한다.

```
------
>>> C = [0, 0]
>>> D = [1, 2]
>>> from scipy.spatial.distance import pdist  <-- (1)
>>> pdist((C, D))  <-- (2)
```

```
array([ 2.23606798])  <-- (3)
>>>
------
```

`scipy.spatial.distance` 모듈로부터 `pdist`를 호출하여 (1), 측정법을 지정하지 않고 `C`와 `D` 간의 거리를 계산하면 (2), 기본값인 유클리드 거리를 적용하여 그 결과를 출력한다 (3).

보통 유클리드 거리는 R 패키지나 MATLAB®[1] 등의 여러 계산 도구의 다양한 거리 관련 연산 (hierarchical clustering, K-means clustering, etc.)에서 기본 측정법으로 사용된다.

또한, `scipy` 모듈에 관하여 더 자세하게 알고 싶다면 scipy.org[2]를 참조해 보기 바란다. 거리 계산에서 짝으로 구성된 자료 간의 거리를 계산하는 `pdist()` 외에도 다른 구조의 자료에 대해 계산을 수행하는 기능에 대해서도 확인할 수 있다.

고주온(필명) (http://bioprofiler.tistory.com)
IBM-XT시절부터 개인용 컴퓨터를 사용하였으나, 강산이 변한 지금도 어제 코딩 한 내용을 오늘 기억하지 못하는 자유인. 박사학위는 분자유전학 분야로 받았으며, 물리학과 화학에 관심만 있음. 현재 대학 교수로 재직 중.

고주온 (2016). [초심자를 위한 생물학+정보학] _37_다양한 거리 계산.
Available from http://www.ibric.org/myboard/read.php?Board=news&id=272298 (May.20, 2016)

Email: member@ibric.org

[1] MathWorks®에서 발매하는 전산 도구이다.
[2] http://scipy.org