인공지능 수학 개론

(저자 김종락)

(임시적인) 목록

Chapter 1. 파이썬 소개: 수식 중심으로

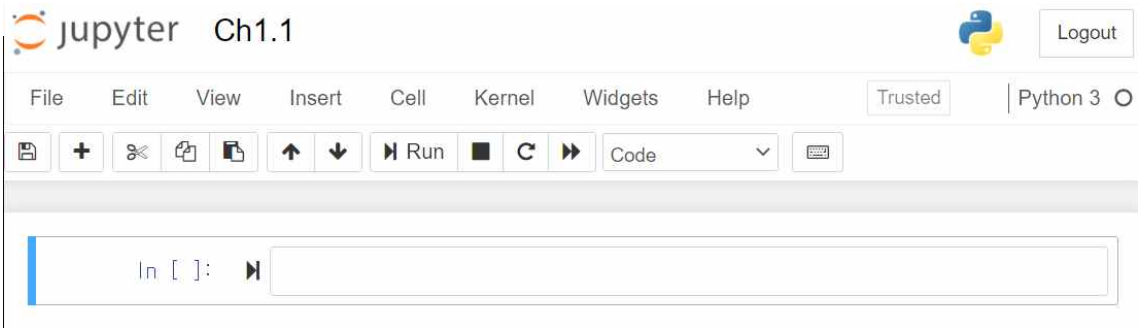
파이썬은 R과 함께 데이터 분석에서 널리 쓰이는 오픈 소스 언어이다. R은 통계적인 면에서 강점이 있고 파이썬은 좀 더 고급 데이터 분석을 할 때 유용한 장점이 있다. 여기에서는 파이썬 위주로 수학식을 실습하고자 한다.

파이썬 3이상의 버전을 쓰도록 한다. 이를 직접 자신의 노트북에 설치하여 주피터 노트북(Jupyter Notebook)에서 작업을 하거나 설치가 어려운 사람은 구글 CoLab에서 로그인해서 진행하면 된다. CoLab의 주소는 https://colab.research.google.com/ 이다. CoLab에는 Tensorflow도 설치되어 있어 직접 설치하는 수고를 피할 수 있다. 필자는 Jupyter Notebook을 주로 사용하였다. 명령어만 넣으면 다음 단계에 오류가 있는지 확인이 되기 때문에 실수를 줄일 수 있다. 또한 자신의 로컬 파일들을 읽고 저장하기 용의하다. 다만 용량이 큰 프로젝트를 할 경우 제약이 있다. 자신의 환경에 맞게 진행을 하면 된다.

파이썬 설치는 https://www.python.org/에 가서 다운받거나 다른 인터넷 사이트들이 있으니 생략하도록 한다. 대신 최신 파이썬(현재 3.10)이 항상 좋은 것이 아니다. 우리가 필요로하는 scipy, numpy, Tensorflow 등은 파이썬 3.6에서 무리없이 작동이 되기 때문이다. 대신 64비트용 설치 파일을 권장한다.

주피터 노트북으로 시작하기

주피터 노트북을 성공적으로 설치한 후 New 페이지를 열면 아래와 같은 화면이 나온다. 첫 파일명은 Untitled이며 확장자는 ipynb이다. 앞의 i는 interactive하다는 의미에서 나온 것이다. 아래 화면에서는 편이상 Ch1.1로 지정하였다.



이 빈칸에 1+1을 입력한 후 그냥 Enter를 치면 줄만 아래로 바뀔 분 결괏값이 나오지는 않는다. 이 기능은 여러 개의 명령어를 한꺼번에 넣고 그 결과를 알고 싶을 때 유용하다. 하나 이상의 명령어를 실행시키는 간단한 방법은 Shift+Enter를 치는 것이다. 그러면 아래와 같이 결괏값 2를 출력하고 새로운 입력값 라인이 형성된다.

수식 뒷 부분에 등호(=)는 넣지 않는다. 대신 변수를 지정할 경우는 =를 넣는다. 즉 a=1, b=2라고 하고 c=a+b라고 할 때, c의 값을 구하고자 할 경우 아래와 같이 대입한다. 각 라인 끝에는 코마를 붙이지 않는다.

```
### 변수값 대입
a=1
b=2
c=a+b
c
```

```
3
```

입력값 사이에 코마(,)를 넣을 경우 c의 값은 tuple이라는 type의 (1,2)가 나오니 주

의해야 한다.

```
### 변수사이 코마
a=1,
b=2,
c=a+b
c
```

```
(1, 2)
```

```
type(c)
```

```
tuple
```

덧셈과 마찬가지로 뺄셈은 '-' 기호로, 곱셈은 '*' 기호로, 나눗셈은 '/' 기호를 이용하여 자연스럽게 계산하면 된다.

```
### 뺄셈
a = 5-3
print(a)
```

```
2
```

```
### 곱셈
b = 5*3
print(b)
```

```
15
```

```
### 나눗셈
c = 5/3
print(c)
```

```
1.6666666666666667
```

곱셈 기호를 두 번 연속 즉 '**'라고 쓸 경우 지수가 된다. 나눗셈의 경우 몫(quotient)을 구하고 싶으면 '//'를 나머지(remainder)를 구하고 싶으면 '%'를 사용한다.

```
### 지수(power)
d = 5**3
print(d)
```

```
125
```

```
### 몫 구하기
e = 5 // 3
print(e)
```

```
1
```

```
### 나머지 구하기
f = 5 % 3
print(f)
```

```
2
```

지금까지 파이썬의 명령어를 이용하여 사칙연산을 계산하는 방법을 배웠다. 이 정도면 파이썬으로 전자계산기를 대체할 수 있다는 것을 알 수 있다.

print의 기능을 좀 더 유연하여 이용하여 결괏값을 좀 더 보기 좋게 만들 수 있다. print 안에는 문자열 또는 수식을 넣을 수 있는데, 문자열을 넣는 경우 “ ”를 두어야 한다. 수나 수식을 넣을 경우는 “ ”를 두지 않는다.

```
### print 명령어의 형식
print("Hello, there. I like 3") # put " " inside of characters
```

Run this cell

```
Hello, there. I like 3
```

```
print(2+3)
```

```
5
```

```
print("2+3 =", 2+3)
```

```
2+3 = 5
```

```
print("2^3 =", 2**3)
```

```
2^3 = 8
```

예제로서, “googol은 10의 100제곱을 의미합니다.”를 인쇄하고 그 실제값을 출력값으로 나타내보자.

```
print("googol은 10의 100제곱을 의미합니다,", 10**100)
```

```
googol은 10의 100제곱을 의미합니다, 10000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
```

자료구조

파이썬의 자료구조는 크게 리스트(list), 튜플(tuple), 딕셔너리(dictionary)로 이루어져 있다.

● 리스트

리스트는 수학에서 집합과는 약간 다른 개념이다. 집합은 나열된 순서에 상관이 없으나 리스트는 순서가 중요하다. 예를 들어, 1,2,3,4,5를 list_1이라는 변수에 넣으려고 한다. 이때 순서를 고려하여 list_1 = [1, 2, 3, 4, 5]라고 치면 된다. list_1[0]은 1을 의미하고 list_1[4]는 5를 의미한다. list_1 뒤에 숫자를 추가하고 싶으면 list_1.append(숫자)를 쓰면 된다. 만약 특정 위치에 특정 숫자를 넣고자 하면 list_1.insert(위치, 숫자)라고 쓰면 된다. 리스트에 나열된 수들을 반대로 쓰고 싶으면 list_1.reverse()라고 쓰면 된다. 이때 ()안은 빈 칸으로 둔다. 특정 숫자를 지우고 싶으면 list_1.remove(숫자)라고 쓰고 숫자들을 작은 수부터 큰 수로 정렬하고 싶으면 list_1.sort()라고 쓰면 된다.

```
### 리스트란?
list_1 = [1, 2, 3, 4, 5]
list_1.append(6)  # 맨 뒤에 6 추가
print(list_1)
print(list_1[0])  # 첫 번째 위치 원소
print(list_1[1])  # 두 번째 위치 원소

list_1.insert(0, 7) # 맨 앞(0의 위치)에 7 넣기
print(list_1)
list_1.reverse()
print(list_1)

list_1.remove(2)  # 숫자 2 제거하기
print(list_1)

list_1.sort()  # 정렬하기
print(list_1)
```

```
[1, 2, 3, 4, 5, 6]
1
2
[7, 1, 2, 3, 4, 5, 6]
[6, 5, 4, 3, 2, 1, 7]
[6, 5, 4, 3, 1, 7]
[1, 3, 4, 5, 6, 7]
```

리스트에 있는 수들 중 일부를 불러내고 싶을 땐 ':' 기호를 사용한다. 예를 들어 리스트 list_2 = [2,3,5,7,11]이라고 할 때 list_2의 처음 3개의 수를 출력하고 싶으면 list_2[0:3]이라고 치면 된다. 그러면 2,3,5가 출력된다. [0:3]의 의미는 0번째부터 3번째 이전까지의 수를 의미한다. 즉 index 3에 해당하는 수는 출력하지 않는다.

어떤 특정 위치를 포함한 모든 수를 출력하고 싶을 경우에도 :을 사용할 수 있다.

```
list_2[1:] #1번째 이상에 있는 모든 수 출력
```

```
[3, 5, 7, 11]
```

```
### 작은 소수들의 모임
list_2 =[2, 3, 5, 7,11]
print(list_2)
```

```
[2, 3, 5, 7, 11]
```

```
list_2[0:3] #0~2번째 있는 수들이 출력된다.
```

```
[2, 3, 5]
```

● 튜플

이제는 튜플(tuple)에 대해서 간단히 설명해 보자. 튜플은 리스트와 마찬가지로 수 또는 문자들을 순서대로 나열한다. 튜플과 리스트의 가장 큰 차이점은 튜플의 값들은 변경이 되지 않는다는 것이다. 튜플이 수로 이루어졌을 경우 길이가 n인 벡터로 이해하면 된다. 실제로 벡터를 구성하는 값들은 없애거나 추가할 수 없다. 두 개의 튜플은 더할 수 있으나 벡터의 합과는 다소 다름에 주의해야 할 것이다. 또한 하나의 튜플에 자연수를 곱할 수도 있다. 아래 예제들로 이를 설명해 보자.

```
# 튜플(tuple)
c = (1, 3, 5, 7, 9)
d = (2, 4, 6, 8,10)
print(c)
print(d)
c[0]
```

```
(1, 3, 5, 7, 9)
(2, 4, 6, 8, 10)

1
```

c는 1,3,5,7,9로 이루어진 튜플이고 d는 2,4,6,8,10으로 이루어진 튜플이다. c의 첫 번째 원소를 알고 싶으면 ‘c[0]’ 라고 쓰면 된다. 이는 리스트와 같은 형식이니 새로울 것은 없다. 튜플 c의 길이를 알고 싶으면 len(c)를 하면 된다.

두 개의 튜플을 더하면 어떨까? 이 경우는 첫 번째 튜플의 값들 뒤에 두 번째 튜플의 값들이 연결되어 새로운 튜플이 생성된다. 위의 예제로 하면 c+d=(1,3,5,7,9,2,4,6,8,10)이 된다.

```
c[0]
```

```
1
```

```
### 튜플의 길이는 len()
len(c)
```

```
5
```

```
### 두 개의 튜플을 + 하면 옆으로 붙는다.
c+d
```

```
(1, 3, 5, 7, 9, 2, 4, 6, 8, 10)
```

따라서 튜플사이의 기호 '+'는 덧셈을 의미하는 것이 아님에 주의하도록 하자. 한편 하나의 튜플에 '*'를 사용할 수도 있다. 예를 들어 c*3라고 쓰면 이는 c+c+c를 의미한다. 즉 c의 원소들이 3번씩 반복된다.

```
### 여러개의 튜플에 +을 붙일 수 있다.
c+c+c
```

```
(1, 3, 5, 7, 9, 1, 3, 5, 7, 9, 1, 3, 5, 7, 9)
```

```
### 튜플에 '* 숫자'를 하면 그 만큼 반복된다.
c*3
```

```
(1, 3, 5, 7, 9, 1, 3, 5, 7, 9, 1, 3, 5, 7, 9)
```

```
### 튜플의 비교
print("c+c+c와 c*3가 같은가?", c+c+c == c*3)
```

```
c+c+c와 c*3가 같은가? True
```

튜플은 숫자의 일반화이기 때문에 크기를 비교할 수 있다. 크기의 기준은 첫 번째 성분들을 비교하여 더 큰 것이 크다. 만일 이것이 같으면 두 번째 성분들을 비교하면 된다. 여기에서 크기가 결정이 나면 나머지 성분들은 비교할 필요가 없다. 이런 것을 사전식 크기(dictionary order)라고 한다. 영어사전을 보면 단어들의 배열은 a로 시작하는 단어들이 맨 먼저 오고 그런 단어들 사이에는 두 번째 알파벳이 가장 작은 것이 더 먼저 나오는 것과 같은 원리이다.

예를 들어 tuple1=(1,3,5)라고 하고 tuple2=(1,4,3)라고 하면 어떤 것이 더 클까? 아

니면 비교 불가능할까? 정답은 tuple2가 더 크다. 첫 번째 성분은 둘다 1이지만, 두 번째 성분은 3<4이므로 tuple2가 더 크다. 세 번째 성분은 5>3이지만 이미 2번째 성분에서 크기가 결정되었으므로 고려하지 않아도 된다.

```
tuple1 =(1,3,5)
tuple2 =(1,4,3)
print("list2 is greater than list1:", list2 > list1)
```

```
list2 is greater than list1: True
```

튜플의 값들은 변경이 되지 않지만 값의 위치라든지 값이 몇 개가 있는지를 세는 것은 가능하다. 튜플 값의 위치는 '튜플.index(값)'의 명령어를 사용한다. tuple1.index(5)라고 하면 5가 있는 index를 구하라는 것이므로 2가 된다. tuple3=(3,3,4,4,4,5)라고 할 때, 4이 몇 번 나오는지 구하고자 할 경우는 '튜플.count(값)'라는 명령어를 사용한다. 따라서 tuple3.count(4)은 3이 된다.

```
### 튜플 값의 위치와 갯수 구하기
tuple1 =(1,3,5)
tuple3 = (3,3,4,4,4,5)
print("tuple1에서 5의 위치는?", tuple1.index(5))
print("tuple3에서 4의 빈도수는?", tuple3.count(4))
```

```
tuple1에서 5의 위치는? 2
tuple3에서 4의 빈도수는? 3
```

튜플과 리스트가 유사하다보니 서로 변환이 가능하다. 튜플을 리스트로 변환하는 명령어는 'list((튜플명))'이다. 튜플명에 '( )'를 넣는 것에 주의하자. 마찬가지로 리스트를 튜플로 변화하는 명령어는 'tuple((리스트명))'이다.

```
### tuple을 리스트로 변환하기
print("tuple1의 list형태:",list((tuple1))  )
```

```
tuple1의 list형태: [1, 3, 5]
```

```
### list를 tuple로 변환하기
list_1 = [1, 2, 3, 4, 5]
print("list의 tuple 형태:",tuple((list_1))  )
```

```
list의 tuple 형태: (1, 2, 3, 4, 5)
```

● 딕셔너리

딕셔너리(dictionary)는 key와 그에 대응하는 value가 하나의 쌍이 되고 이 쌍들을 나열한 것들의 모임이다. 딕셔너리는 리스트와 튜플과 구별하기 위하여 중괄호 '{ }'를 사용한다. 형식은 {key1:value1, key2:value2, ...}이다. key값은 숫자 또는 문자가 될 수 있다. 중복이 되지 말아야 한다. 좀 더 일반적으로는 key값은 튜플을 사용할 수도 있다. 대신 list는 key 값이 될 수 없으니 조심하도록 하자.

예를 들어 AI과목이라는 딕셔너리에 인공지능수학, 머신러닝, 딥러닝을 넣고 싶고 각각의 value를 1, 2, 3이라는 key값을 이용한다면 AI과목 = {1:"인공지능수학", 2:"머신러닝", 3:"딥러닝"}라고 입력하면 된다. key 값 1을 이용하여 인공지능수학이라는 value를 출력할 수 있다.

```
### dictionary형태
AI과목 = {1:"인공지능수학", 2:"머신러닝", 3:"딥러닝"}
print("AI 과목", AI과목)
```

```
AI 과목 {1: '인공지능수학', 2: '머신러닝', 3: '딥러닝'}
```

```
### dictionary에서 value를 출력하려면 key값 부른다
AI과목[1]
```

```
'인공지능수학'
```

value는 리스트가 될 수 있다. '성적'이라는 변수에 인공지능수학과 머신러닝을 A학점, 딥러닝을 B학점이라고 할 때 이를 딕셔너리로 표현하면 다음과 같다.

```
### value를 리스트로 저장한 후 value 출력하기
성적 = {"A":["인공지능수학", "머신러닝"], "B":["딥러닝"]}
성적["A"]
```

```
['인공지능수학', '머신러닝']
```

get함수를 써서 value를 출력할 수 있다.

```
### 또는 .get을 쓸 수 있다.
성적.get("A")
```

```
['인공지능수학', '머신러닝']
```

특정 value뿐만 아니라 모든 value를 출력하고 싶으면 '변수.values()'라고 쓰고 모든 key를 출력하고 싶으면 '변수.keys()'라고 쓰면 된다.

```
### dictionary의 values만 출력하고자 할 때
성적.values()
```

```
dict_values([['인공지능수학', '머신러닝'], ['딥러닝']])
```

```
### dictionary의 keys만 출력하고자 할 때
성적.keys()
```

```
dict_keys(['A', 'B'])
```

리스트와 마찬가지로 딕셔너리도 key:value 쌍을 추가하거나 삭제할 수 있다. 추가하고자 할 경우는 '변수[새로운 key] = value'의 형태로 진행하면 된다. 삭제하고자 할 경우는 'del 변수[key]'를 하면 해당하는 key:value가 삭제된다.

```
### dictionary에 key:value를 추가하고자 할 때
성적 = {"A":["인공지능수학", "머신러닝"], "B":["딥러닝"]}
성적["C"] ="확률론"
새성적 = 성적
print("새 성적 리스트", 새성적)
```

```
새 성적 리스트 {'A': ['인공지능수학', '머신러닝'], 'B': ['딥러닝'],
'C': '확률론'}
```

```
### dictionary에 key:value를 삭제하고자 할 때 'del 변수[key]'형태
성적 = {"A":["인공지능수학", "머신러닝"], "B":["딥러닝"]}
del 성적["B"]
성적
```

```
{'A': ['인공지능수학', '머신러닝']}
```

또한 특정 key의 value만 수정할 수도 있다. 이런 경우 '변수(key) = 새 value'라고 쓰면 된다. 예를 들어 성적이라는 변수에서 B를 맞은 딥러닝을 프로그래밍으로 대체할 수 있다.

```
### dictionary에 key값의 value를 수정하고자 할 때
성적 = {"A":["인공지능수학", "머신러닝"], "B":["딥러닝"]}
성적["B"] = "프로그래밍"
성적
```

```
{'A': ['인공지능수학', '머신러닝'], 'B': '프로그래밍'}
```

마지막으로 딕셔너리 변수의 모든 key:value를 다 삭제하려면 '변수.clear()'형태를 사용하면 된다.

```
### dictionary의 값을 모두 삭제하고 싶을 때
성적.clear()
성적
```

```
{}
```

● 세트

세트(set)는 수학에서 말하는 집합을 의미한다. 따라서 리스트나 튜플과는 달리 세트의 원소들 사이에는 순서가 없다. 또한 원소들이 중복되지 않는다. 두 집합사이에 교집합, 합집합, 차집합 등의 연산이 존재한다.

```
### set 예제
set1 = {1, 3, 5}
set2 = {1, 1, 3, 5}
print("set1과 set2는 같은가?", set1 == set2 )
```

```
set1과 set2는 같은가? True
```

또한 리스트로부터 집합을 정의할 수 있다.

```
### list로부터 set 정의하기
set3 = set([1,1,2,3])
set3
```

```
{1, 2, 3}
```

교집합은 '변수1.intersection(변수2)' 또는 '변수1 & 변수2'로 표현가능하고 합집합은 '변수1.union(변수2)' 또는 '변수1 | 변수 2'로 표현할 수 있다.

```
### 두 sets의 교집합 첫번째 방법
### set1 = {1, 3, 5}, set3 = {1, 2, 3}
set1.intersection(set3)
```

```
{1, 3}
```

```
### 두 sets의 교집합 두번째 방법
set1 & set3
```

```
{1, 3}
```

```
### 두 sets의 합집합 첫번째 방법
set1.union(set3)
```

```
{1, 2, 3, 5}
```

```
### 두 sets의 합집합 두번째 방법
set1 | set3
```

```
{1, 2, 3, 5}
```

두 집합 A, B의 차집합은 A의 원소 중에 B에 있지 않는 것들의 모임으로 수학적으로는 'A-B'로 표시한다. 실제로는 A-(A&B)에 해당한다. 또한 A, B의 합집합에서 교집합을 뺀 집합은 (A | B)-(A&B)에 해당하고 'A ^ B'로 표시한다.

```
### 두 sets A,B의 차집합= A-(A &B)
set1 - set3
```

```
{5}
```

```
### 두 sets A,B의 합집합에서 교집합을 뺀 것
set1 ^ set3
```

```
{2, 5}
```

리스트나 딕셔너리처럼 원소를 추가할 수 있는데 원소 하나를 추가할 경우는 .add()라고 쓰고 여러 개의 원소를 추가할 경우는 .update()라고 쓴다. 원소 하나를 제거할 때는 .remove()라고 쓴다.

```
### set에 원소 추가하기 .add()
set1 ={1, 3, 5}
set1.add(7)
print("set1=",set1)  # 변수 set1는 같으나 값은 변하였음.
```

```
set1= {1, 3, 5, 7}
```

```
### set에 2개 이상의 원소 추가하기 .update()
set1 ={1, 3, 5}
set1.update([7,9,11])
print("set1 update =",set1)
```

```
set1 update = {1, 3, 5, 7, 9, 11}
```

```
### set의 원소 제거하기 .remove()
set1 ={1, 3, 5}
set1.remove(1)   # list로는 안됨.
print("set1 remove =",set1)
```

```
set1 remove = {3, 5}
```

지금까지 1장에서는 파이썬의 기본 데이터 형태인 리스트, 튜플, 딕셔너리, 세트에 대하여 간단히 살펴 보았다. 2장에서는 선형대수에 나오는 기본적인 수식들을 파이썬으로 표현하는 것을 배우도록 할 것이다.