

Day5 Presentation Study

Logging ... p.2

공유기 ... p.9

KNN ... p. 20

Closure 사용 이유 ... p.31

Configuration ... p.46

logging

logging vs print

- 둘 다 우리가 원하는 내용을 어떠한 형태로 남긴다는 점에서는 비슷하다
- 출력에 대해 조금 더 구체적으로 설정할 필요가 있다면 로깅이 좋음 (특정 상황 부여, 포맷 저장)

로깅 모듈 사용하기

```
In [10]: yourlogger = logging.getLogger("your")
yourlogger.setLevel(logging.CRITICAL)

yourlogger.critical("server start!!!")
```

```
CRITICAL:your:server start!!!
```

```
import logging

mylogger = logging.getLogger("my")
mylogger.setLevel(logging.INFO)

formatter = logging.Formatter('%(asctime)s - %(name)s - %(levelname)s - %(message)s')

stream_hander = logging.StreamHandler()
stream_hander.setFormatter(formatter)
mylogger.addHandler(stream_hander)

file_handler = logging.FileHandler('my.log')
mylogger.addHandler(file_handler)

mylogger.info("server start!!!")
```

```
2021-01-25 00:03:28,424 - my - INFO - server start!!!
2021-01-25 00:03:28,424 - my - INFO - server start!!!
2021-01-25 00:03:28,424 - my - INFO - server start!!!
```

```
1 server start!!!
2 server start!!!
3 server start!!!
4 server start!!!
5 server start!!!
6 server start!!!
7 server start!!!
8 server start!!!
9 server start!!!
10 server start!!!
11 server start!!!
12 server start!!!
13 server start!!!
... . . . . .
```

Root 로깅을 만들고 싶다면?

```
logging.error('에러났어')  
ERROR:root:에러났어
```

```
In [19]: log=logging.getLogger()
```

```
In [20]: log.setLevel(logging.DEBUG)
```

```
In [22]: logging.info('성공!!')
```

```
INFO:root:성공!!
```

configparser

```
1 [loggers]
2 keys = root
3
4 [handlers]
5 keys = consoleHandler
6
7 [formatters]
8 keys=simpleFormatter
9
10 [logger_root]
11 level=DEBUG
12 handlers = consoleHandler
13
14 [handler_consoleHandler]
15 class=StreamHandler
16 level=DEBUG
17 formatter = simpleFormatter
18 args=(sys.stdout,)
19
20 [formatter_simpleFormatter]
21 format = %(asctime)s %(levelname)s [%(name)s] [%(filename)s:%(lineno)d] - %(message)s
```

```
In [17]: import logging
import logging.config

logging.config.fileConfig("logging.conf")

In [18]: print('a')
logging.error('hi')
print('b')

a
2021-01-25 13:48:14,844 ERROR [root] [<ipython-input-18-9bcfba2ccc62>:2] - hi
b
```

조금 더 복잡한 config

```
1 [formatters]
2 keys=simple,complex
3
4 [formatter_simple]
5 format=[%(name)s] %(message)s
6
7 [formatter_complex]
8 format=%(asctime)s %(levelname)s [%(name)s] [%(filename)s:%(lineno)d] - %(message)s
9
10 [handlers]
11 keys=console,file
12
13 [handler_console]
14 class=StreamHandler
15 args=(sys.stdout,)
16 formatter=simple
17 level=DEBUG
18
19 [handler_file]
20 class=FileHandler
21 args=("error.log",)
22 formatter=complex
23 level=ERROR
24
25 [loggers]
26 keys=root,parent,child
27
28 [logger_root]
29 level=WARNING
30 handlers=console,file
31
32 [logger_parent]
33 qualname=parent
34 level=INFO
35 handlers=
36
37 [logger_child]
38 qualname=parent.child
39 level=DEBUG
40 handlers=
```

```
[28]: import logging
import logging.config
logging.config.fileConfig('logging.conf')
```

```
[34]: parent_logger = logging.getLogger('parent')
parent_logger.critical('haha')
```

```
[parent] haha
```

```
1 | 2021-01-25 14:01:46,963 CRITICAL [parent] [<ipython-input-33-97f477bb0a32>:2] - haha
```

수업자료

```
[loggers]
keys=root

[handlers]
keys=consoleHandler

[formatters]
keys=simpleFormatter

[logger_root]
level=DEBUG
handlers=consoleHandler

[handler_consoleHandler]
class=StreamHandler
level=DEBUG
formatter=simpleFormatter
args=(sys.stdout,)
```

공유기

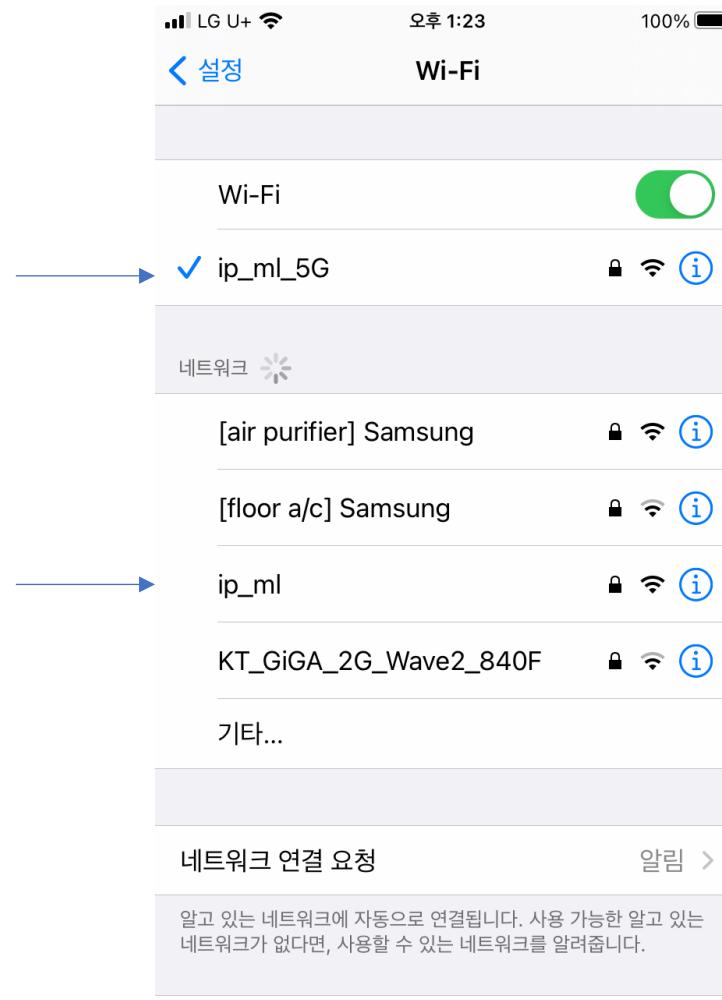
유선 공유기, 무선 공유기, 무선 랜카드

- 유선 공유기 = wi-fi 안됨
 - 보통.. 데스크탑에
- 무선 공유기 = 유선 안됨
 - Wi-fi 생성
- 무선 랜카드 = wi-fi를 잡을 수 있게 해주는 부품
 - 주로 데스크탑에서 사용
 - 노트북은 무선 랜카드가 탑재되어 있음

공유기 스펙

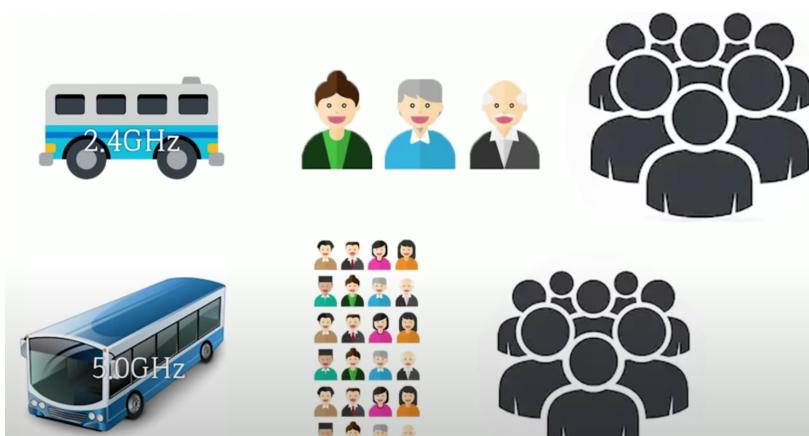
- 802.11n
 - 옛날 방식
 - n = 2.4Ghz 주파수 지원
 - 싱글밴드
- 802.11ac
 - 요즘 방식
 - ac = 5Ghz 주파수 지원
 - 2.4, 5Ghz 둘 다 지원
 - 듀얼 밴드: 둘 다 지원

Ex) 트라이 밴드는 신호가 3개 (2.4Ghz/5.0Ghz/5.0Ghz)



공유기 스펙

- 2.4Ghz = 최대 300Mbps
 - 물체를 잘 통과 -> 넓은 범위
- 5.0Ghz = 최대 1600Mbps(제품 대비 따라 지원이 낮을 수 있음)
 - 물체를 잘 통과 못함 -> 좁은 범위



무조건 5.0Ghz?

- 방 과 장해물(다른 방)이 있다면 5Ghz는 수신이 불안정
 - 한 공간, 근접한 공간에서 사용하는 것이 좋음

정리

느림
많음
넓음
쌈

속도
방해
범위
가격

빠름
적음
좁음
비쌈



dBi?

- 안테나 발신 범위의 형태
 - 최근에는 4
 - dBi가 높아질수록 발신 형태가 원형에서 길게 늘어진 타원으로 바뀜
 - 높다고 넓어지는 그런개념은 아닌듯
 - 잘 안되면 방향을 틀어보면..?

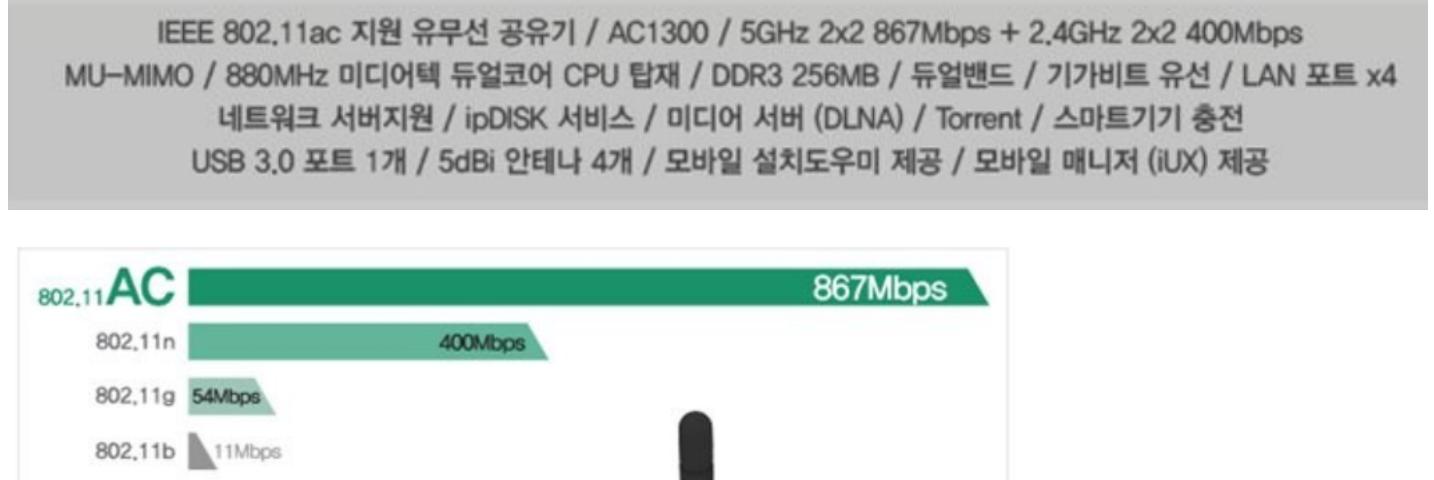
공유기의 CPU

- 리얼텍
- 미디어텍
 - 가성비 좋아서 이거 많이 씀
- 브로드컴
 - 가장 비싸고 속도가 빠름
 - 발열(=불안정)
 - 전문가용

공유기의 RAM

- 공유기의 RAM은 한번에 다중접속 허용
 - RAM이 높을수록 많은 동시 접속 허용

마지막 정리 – 앞광고(x) 뒷광고(x)



- MU-MIMO
- 여러 대의 디바이스들이 동시에 접속시 신호를 동시에 처리
- 왜 USB3.0? 간이 NAS 지원 - 인터넷 LAN 케이블로 연결하는 외장하드(클라우드)

Reference

- https://www.youtube.com/watch?v=7Kgl3-1FOdQ&ab_channel=%EA%B9%A8%EB%B4%89%EC%B1%84%EB%84%90
- https://www.youtube.com/watch?v=KSFTAupElJ8&ab_channel=%EC%BD%94%EB%94%A9%ED%95%98%EB%8A%94%EA%B1%B0%EB%8B%88
- https://www.youtube.com/watch?v=l9e1t3vinVw&ab_channel=%EC%84%B8%EB%8B%A4%EB%A6%ACTV
https://www.youtube.com/watch?v=Gt6usj5hOZQ&ab_channel=%EC%84%B8%EB%8B%A4%EB%A6%ACTV

KNN Algorithm

K-Nearest Neighbors Algorithm

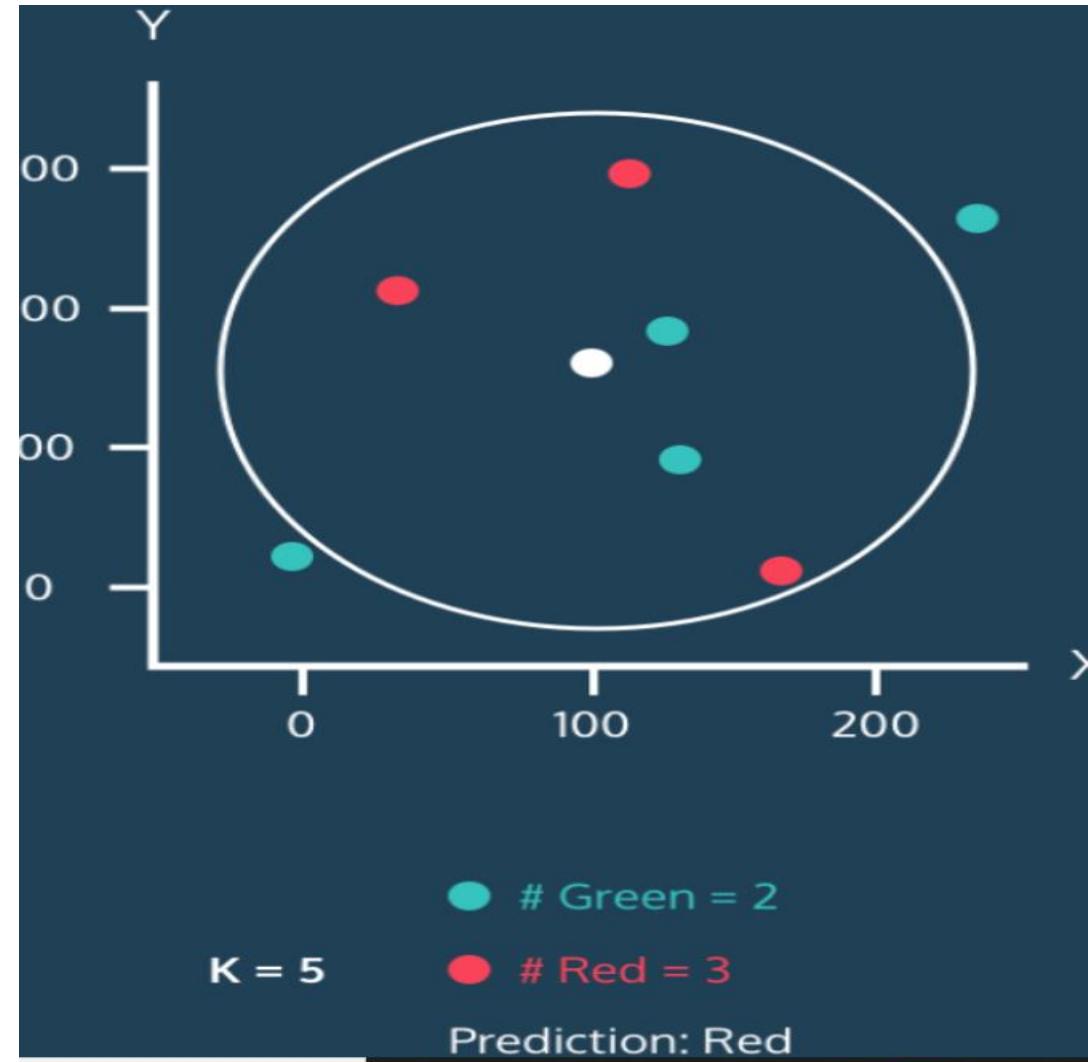
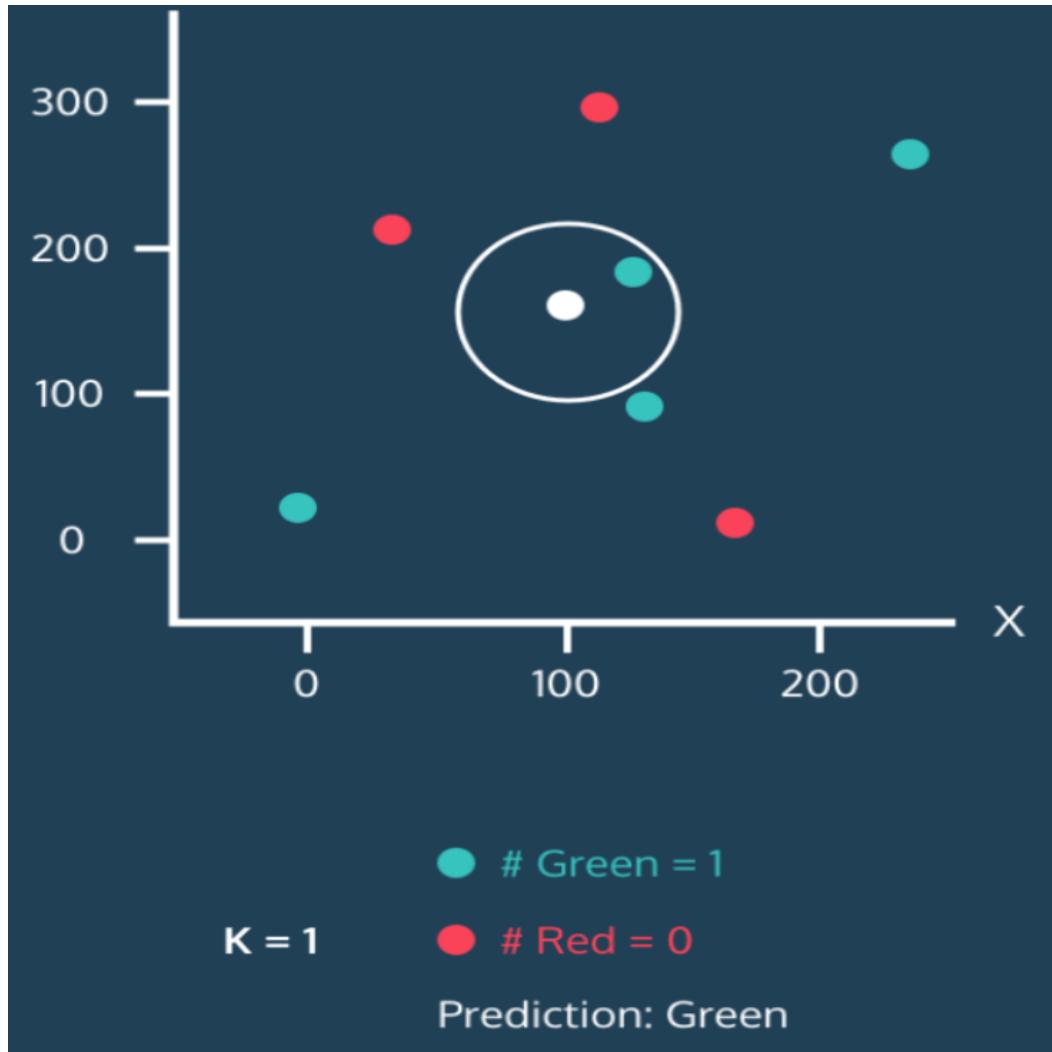
Parameter Tuning

Normalization & scikit-learn

K-Nearest Neighbors Algorithm

- ML 알고리즘에서 분류/회귀(예측) 문제에 사용될 수 있는 알고리즘
- SVM, 의사결정트리 등과 비교연구될 수 있음
- How: test data를 기준으로 sample data point간 거리를 구해서 가장 가깝게 거리가 산출된 상위 k개 데이터의 레이블을 참고함.
- Hyperparameter: K, 거리 측정 방법

K-Nearest Neighbors Algorithm

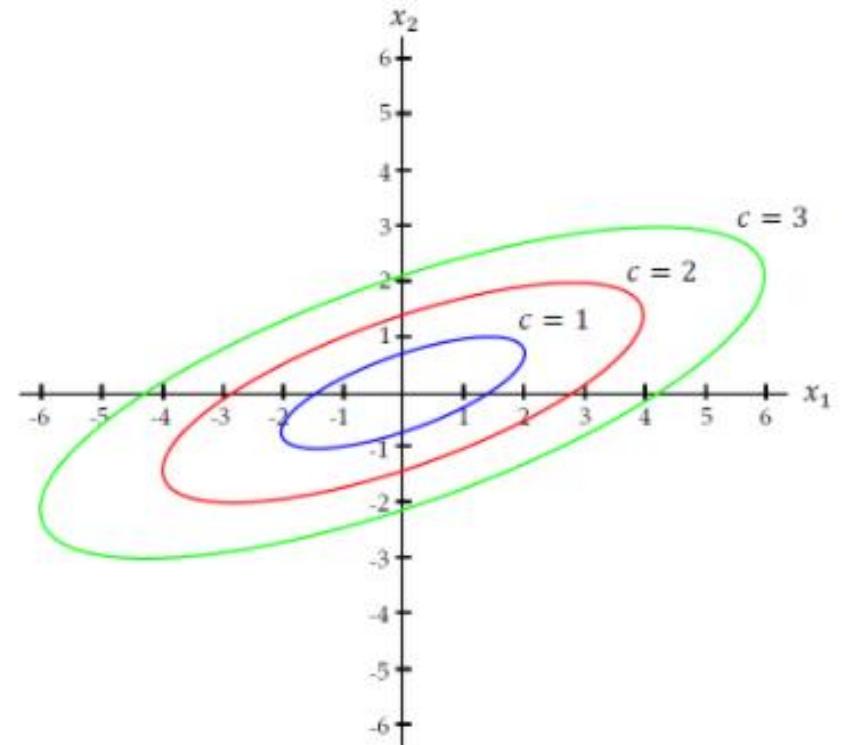


Parameter Tuning

- 거리 지표: Euclidean Distance, Manhattan Distance, 피어슨, spearman Rank Correlation Distance,...
- Mahalanobis Distance: 변수 간 공분산, 상관관계 반영.

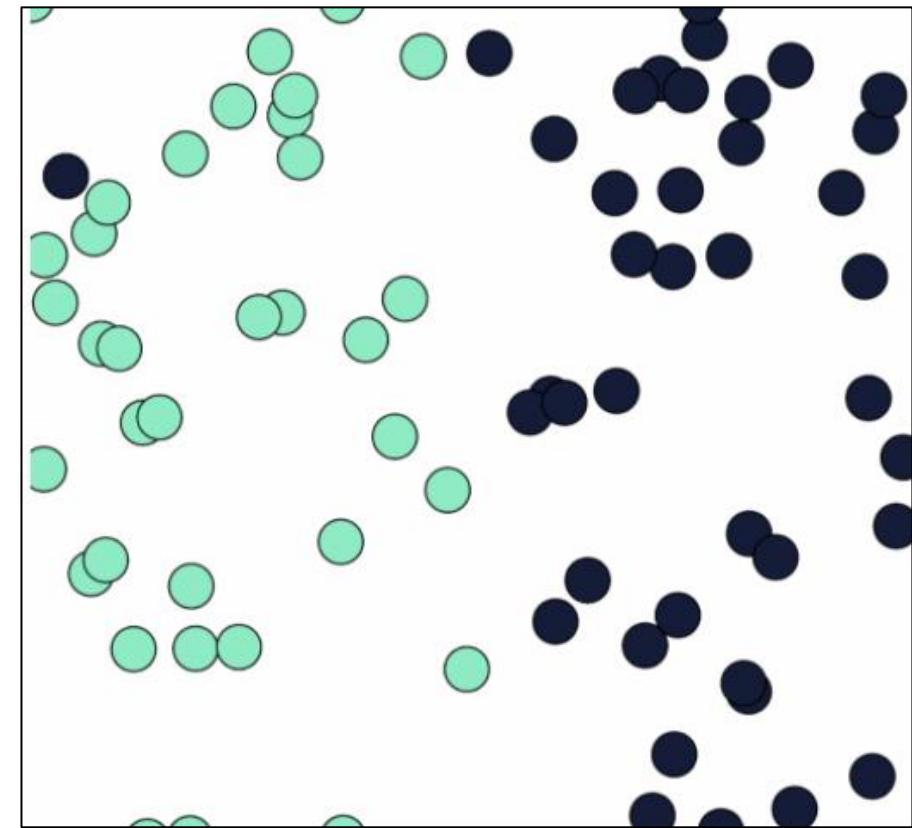
$$d_{Mahalanobis}(X, Y) = \sqrt{(\vec{X} - \vec{Y})^T \Sigma^{-1} (\vec{X} - \vec{Y})}$$

Σ^{-1} = inverse of covariance matrix



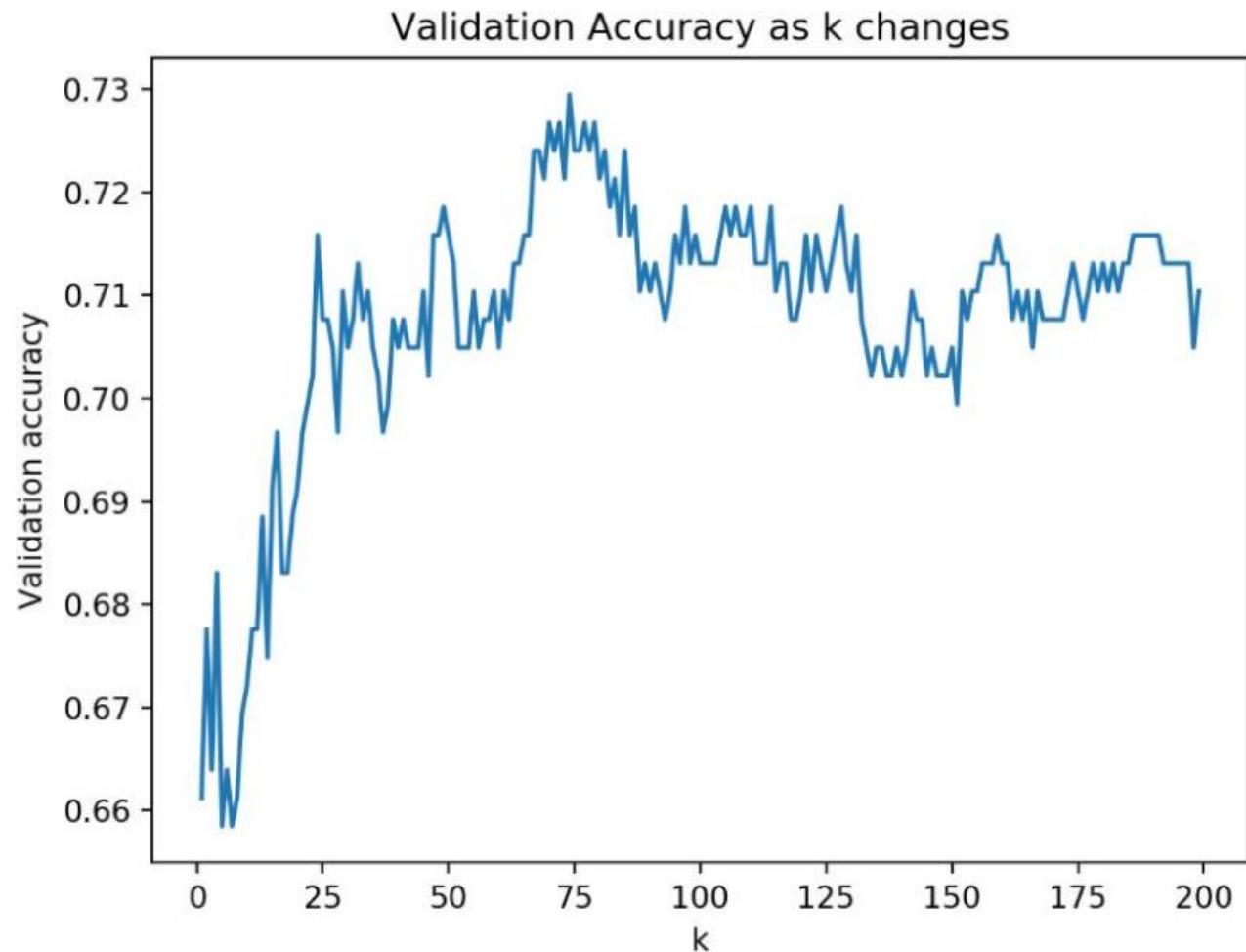
Parameter Tuning

- Tune K: 이상점(Outlier)를 고려해서 최적의 K로 조정할 수 있어야 한다.
→ 옆 그림의 예로, K가 1에 가까운 수일 시, 검은색 이상점에 근접하게 위치한 초록색 표본에 영향을 끼칠 수 있다(오버피팅 문제). 당연하게, 샘플 데이터에 가깝게 큰 수로 K를 지정하면 언더피팅 문제를 일으킨다.

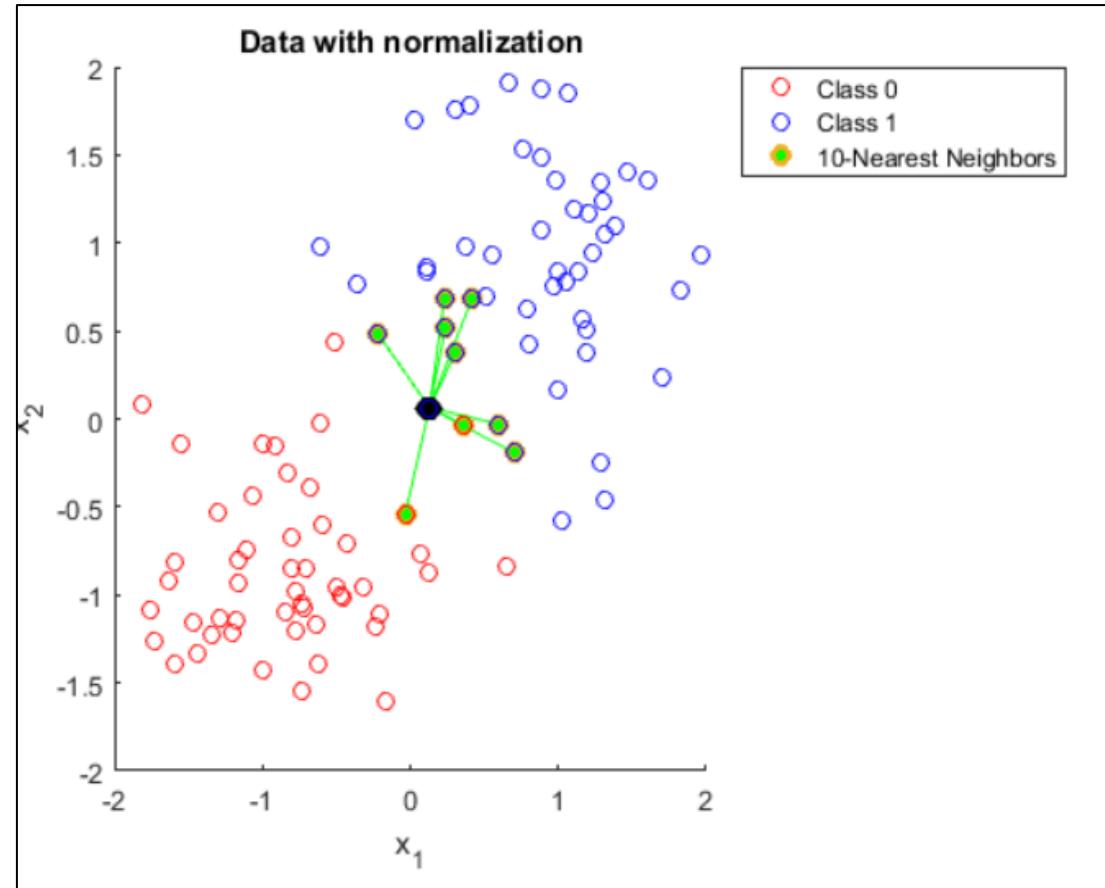
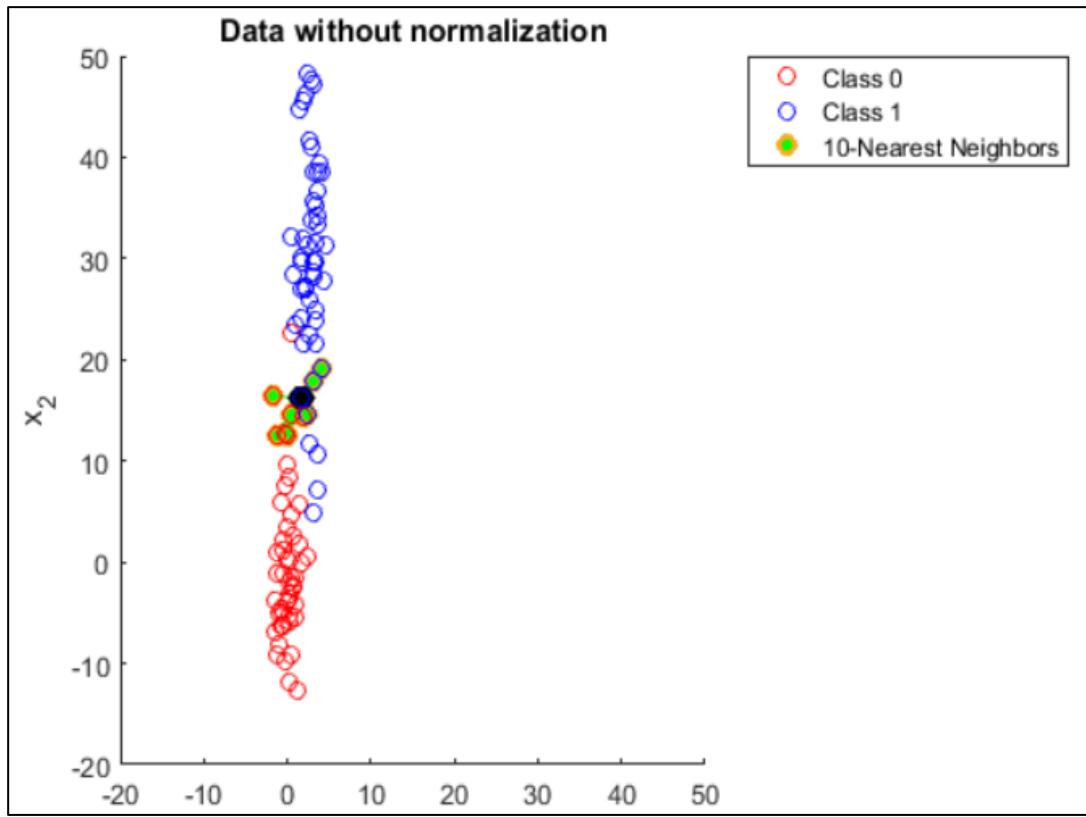


Parameter Tuning

- 일반적으로, 샘플 총 개수의 제곱근 정도로 K를 튜닝하는 것이 권장되나, 옆 사진과 같이 직접 자신이 사용하는 데이터 셋에 동일한 feature extraction Method를 적용하여 K에 따른 validation을 경험적으로 실험 해보는 것이 가장 좋다.



Normalization & Practice



Normalization & Practice

- 0과 1사이의 값으로 표준화.
- 거리를 산출식으로 사용하는
이 알고리즘은 이상점을 최대
한 완화시켜줘야 효과적으로
적용 가능.
→꼭 필요한 preprocessing.

```
def min_max_normalize(lst):  
    normalized = []  
  
    for value in lst:  
        normalized_num = (value - min(lst)) / (max(lst) - min(lst))  
        normalized.append(normalized_num)  
  
    return normalized
```

Normalization & Practice

- 별도의 구현 없이 사용 하려면, scikit-learn 패키지를 사용 할 수 있다.

```
from sklearn.neighbors import KNeighborsClassifier
```

```
classifier = KNeighborsClassifier(n_neighbors = 3)
```

Parameters:

n_neighbors : int, default=5

Number of neighbors to use by default for `kneighbors` queries.

weights : {'uniform', 'distance'} or callable, default='uniform'

weight function used in prediction. Possible values:

- 'uniform' : uniform weights. All points in each neighborhood are weighted equally.
- 'distance' : weight points by the inverse of their distance. in this case, closer neighbors of a query point will have a greater influence than neighbors which are further away.
- [callable] : a user-defined function which accepts an array of distances, and returns an array of the same shape containing the weights.

algorithm : {'auto', 'ball_tree', 'kd_tree', 'brute'}, default='auto'

Algorithm used to compute the nearest neighbors:

- 'ball_tree' will use `BallTree`
- 'kd_tree' will use `KDTree`
- 'brute' will use a brute-force search.
- 'auto' will attempt to decide the most appropriate algorithm based on the values passed to `fit` method.

Note: fitting on sparse input will override the setting of this parameter, using brute force.

leaf_size : int, default=30

Leaf size passed to BallTree or KDTree. This can affect the speed of the construction and query, as well as the memory required to store the tree. The optimal value depends on the nature of the problem.

p : int, default=2

Power parameter for the Minkowski metric. When $p = 1$, this is equivalent to using Manhattan distance ($\| \cdot \|_1$), and

Normalization & Practice

- <https://www.kaggle.com/shrutimechlearn/step-by-step-diabetes-classification-knn-detailed>, 예제 권장!!
- Pros & Cons:
 - 장점은 구현이 굉장히 간단, classifier 모델 관련 코드가 10줄이 넘어가지 않는다.
 - 전처리도 다른 정규화만 따로 해주는데 이것도 min-max 아니면 standard-Scaler normalization에 국한된다.
 - DL보다 ML이 우세할 수 있는 것은 적은 데이터로도 이뤄낼 수 있는 성능인데, KNN은 직관적인 알고리즘에 불구하고 많은 데이터셋을 가지고도 딥러닝보다 우수한 사례를 많이 보인다. (MNIST)
 - 단점은 어느 모델이나 그렇듯이, 경험적인 튜닝 과정이다. 자신이 사용하는 데이터셋에 적합하게 튜닝을 일일이 실험해서 최적화시켜야 한다.

Reference

- <https://ratsgo.github.io/machine%20learning/2017/04/17/KNN/>, 거리 지표와 KNN
- <http://hleecaster.com/ml-knn-concept/>,
<http://hleecaster.com/ml-knn-classifier-example/>, sklearn와 유방암 데이터를 이용한 KNN 개념과 실습
- <https://stats.stackexchange.com/questions/287425/why-do-you-need-to-scale-data-in-knn>, 표준화가 KNN에 끼치는 영향에 대한 의견
- <https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html> 싸이킷런 document

Closure 사용 이유

Closure란

- Inner function을 반환한 함수

```
def outer_func():
    message = 'Hello'
    def inner_func():
        print(message)
    return inner_func
```

```
def outer_func():
    message = 'Hello'
    def inner_func():
        print(message)
    return inner_func
```

Closure

```
def outer_func():
    message = 'Hello'
    def inner_func():
        print(message)
    return inner_func

outer_func()
```

1
2
3
4
5
6

```
def outer_func():
    message = 'Hello'
    def inner_func():
        print(message)
    return inner_func

my_func = outer_func()
my_func()
```

1
2
3
4
5
6

Closure란

```
def outer_func():
    message = 'Hello'
    print("message 선언")
    def inner_func():
        print(message)
    inner_func()

outer_func()
outer_func()
outer_func()
```

```
message 선언
Hello
message 선언
Hello
message 선언
Hello
```

```
def outer_func():
    message = 'Hello'
    print("message 선언")
    def inner_func():
        print(message)
    return inner_func
```

```
my_func =outer_func()
my_func()
my_func()
my_func()
```

```
message 선언
Hello
Hello
Hello
```

Closure 사용 이유

- 여러 번 변수를 선언하지 않고 저장하여 사용하기 때문에
- Closure를 사용하면 비교적 빠름

- 1) 어디에 저장되어 있는가
- 2) 얼마나 빠른가

어디에 저장되어 있는가

```
my_func =outer_func()  
print(dir(my_func))
```

My_func의 namespace를 확인

```
['__annotations__', '__call__', '__class__', '__closure__', __code__, '__defaults__', '__delattr__', '__dict__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__get__', '__getattribute__', '__globals__', '__gt__', '__hash__', '__init__', '__init_subclass__', '__kwd_defaults__', '__le__', '__lt__', '__module__', '__name__', '__ne__', '__new__', '__qualname__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__']
```

```
print(my_func.__closure__)
```

```
(<cell at 0x000001D2D7025BB0: str object at 0x000001D2D712AEB0>,)
```

Cell 객체 : 여러 스코프에서 참조하는 변수의 값을 저장하는 공간

어디에 저장되어 있는가

```
print(dir(my_func.__closure__[0]))
```

```
['__class__', '__delattr__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__gt__', '__hash__', '__init__',  
'__init_subclass__', '__le__', '__lt__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__', '__sizeof__', '__st  
r__', '__subclasshook__', 'cell_contents']
```

```
print(my_func.__closure__[0].cell_contents)
```

```
Hello
```

어디에 저장되어 있는가

```
def outer_func():
    message = 'Hello'
    message_2 = 'Hi'
    x = 1685
    def inner_func():
        print(message)
        print(message_2)
        print(x)
    return inner_func
```

```
my_func =outer_func()
print(my_func.__closure__)
```

```
(<cell at 0x0000023A7A915BB0: str object at 0x0000023A7AA1ADF0>, <cell at 0x0000023A7A915FD0: str object at 0x0000023A7AA1AC70>, <cell at 0x0000023A7A9B5AC0: int object at 0x0000023A7A8B8910>)
```

어디에 저장되어 있는가

```
my_func =outer_func()  
print(my_func.__closure__[0].cell_contents)  
print(my_func.__closure__[1].cell_contents)  
print(my_func.__closure__[2].cell_contents)
```

```
Hello  
Hi  
1685
```

각각의 값 'Hello', 'Hi', 1685는
각 cell의 cell_contents에 저장되어 있다.

어디에 저장되어 있는가

```
def outer_func():
    message = 'Hello'
    message_2 = 'Hi'
    x = 1685
    def inner_func():
        print(message)
    return inner_func

my_func =outer_func()
print(my_func.__closure__)
```

```
(<cell at 0x000001D8903B5BB0: str object at 0x000001D8904BAE30>,)
```

Closure 함수에서 필요로 하는 local 변수만 저장함

얼마나 빠른가

```
import time
cost = []

def outer(cache,x):
    def inner():
        cache.append(x)

    return inner

for t in range(100):
    cache=[]
    start_time = time.time()
    temp = outer(cache, 'hello')
    for i in range(1000000):
        temp()
    end_time = time.time()
    during_time = end_time-start_time
    cost.append(during_time)

print(cache)
print(f'평균 : {sum(cost)/len(cost)}')
print(f'최소 : {min(cost)}')
print(f'최대 : {max(cost)})
```

평균 : 0.2991498398780823
최소 : 0.2822458744049072
최대 : 0.3530547618865967

얼마나 빠른가

```
import time
cost = []

def outer(cache,x):
    def inner():
        cache.append(x)

    inner()

for t in range(100):
    cache=[]
    start_time = time.time()
    for i in range(1000000):
        outer(cache,'hello')
    end_time = time.time()
    during_time = end_time-start_time
    cost.append(during_time)

print(cache)
print(f'평균 : {sum(cost)/len(cost)}')
print(f'최소 : {min(cost)}')
print(f'최대 : {max(cost)}')
```

```
'hello', 'hello', 'hello', 'hello', 'hello', 'hello', 'hello', 'hello', 'hello',
', 'hello', 'hello', 'hello', 'hello', 'hello', 'hello', 'hello', 'hello', 'hello',
'hello', 'hello', 'hello', 'hello', 'hello', 'hello', 'hello', 'hello', 'hello',
'o', 'hello', 'hello', 'hello', 'hello', 'hello', 'hello', 'hello', 'hello', 'hello',
'hello', 'hello', 'hello', 'hello', 'hello', 'hello', 'hello', 'hello', 'hello',
'hello', 'hello', 'hello', 'hello', 'hello', 'hello', 'hello', 'hello', 'hello',
'hello', 'hello', 'hello', 'hello', 'hello', 'hello', 'hello', 'hello', 'hello',
'hello', 'hello', 'hello', 'hello', 'hello', 'hello', 'hello', 'hello', 'hello',
'hello', 'hello', 'hello', 'hello', 'hello', 'hello', 'hello', 'hello', 'hello',
'hello', 'hello', 'hello', 'hello', 'hello', 'hello', 'hello', 'hello', 'hello',
'hello', 'hello', 'hello', 'hello', 'hello', 'hello', 'hello', 'hello', 'hello',
', 'hello', 'hello', 'hello']
```

```
평균 : 0.6120968770980835
최소 : 0.5854344367980957
최대 : 0.6672115325927734
```

Closure를 사용하면 2배 가량 빠름

사용 예제 1

```
def n_squares(n):
    def result(x):
        return n ** x
    return result

three_squares = n_squares(3)
four_squares = n_squares(4)

print('3의 2승 :', three_squares(2))
print('3의 5승 :', three_squares(5))

print('4의 2승 :', four_squares(2))
print('4의 6승 :', four_squares(6))
```

3의 2승 :	9
3의 5승 :	243
4의 2승 :	16
4의 6승 :	4096

사용 예제 2

```
def in_cache():
    cache = []
    def wrapper(n):
        cache.append(n)
        print(cache)
    return wrapper

temp = in_cache()
temp(3)
temp(6)
temp(8)
```

```
[3]
[3, 6]
[3, 6, 8]
```

```
def in_cache(n):
    cache = []
    def wrapper(n):
        cache.append(n)
        print(cache)
    wrapper(n)
```

```
in_cache(3)
in_cache(6)
in_cache(8)
```

```
[3]
[6]
[8]
```

마치 encapsulation 역할을 함

Reference

<https://nachwon.github.io/closure/>

<https://whatisthenext.tistory.com/112>

https://velog.io/@inyong_pang/Python-Nested-Function-2wk42jt94r

<https://shoark7.github.io/programming/python/closure-in-python#3c>

<https://okky.kr/article/285790>

<https://technote.kr/185>

<https://poiemaweb.com/js-closure>

Configuration

파이썬에서 설정값 관리하기

- 빌트인 데이터 구조를 사용한 설정 (.py) → 보안 이슈 생길 수 있음
- 동적 로딩을 통한 설정 → 설정 파일 경로 동적으로 등록, 임포트 가능한 위치에 있을 필요 없음
- 외부 파일을 통한 설정 (.ini, .json, .xml, .yaml 등) → .gitignore에 명시
- 시스템 환경 변수를 사용한 설정 → 셸 스크립트 사용, 노출 위험 적음

configparser

- 프로그램 실행 설정을 파일에 저장(ini, cfg, conf 등)
- Section, Key, Value 값의 형태로 설정된 설정 파일 사용
 - Section: 대괄호 []
 - Key, Value: 딕셔너리 형태 {key : value}
- 설정 파일을 딕셔너리 형태로 처리
- 데이터형에 관계없이 항상 문자열로 저장 → 자료형을 처리하는 getter 함수 제공
 - getboolean(): yes/no, on/off, true/false, 1/0 인식
 - getInt()
 - getfloat()

Example

```
1 import configparser
2
3 # 설정파일 쓰기
4 config = configparser.ConfigParser()
5 config["Test"] = { "str" : "hello", "int" : 123, "float" : 3.14 , "bool1" : True,
6 "bool2" : "yes", "bool3" : 0}
7 with open('ex.ini', 'w') as handle:
8     config.write(handle)
9
10 # 설정파일 읽기
11 config.read('example.ini')
12 print('Section:', config.sections())
13 keys = config["Test"].keys()
14 for key in keys:
15     print("key : " + key + ", value : " + config["Test"][key])
16
17 print(type(config["Test"]["int"]), type(config["Test"].getint("int")))
18 print(type(config["Test"]["float"]), type(config["Test"].getfloat("float")))
19 print(type(config["Test"]["bool1"]), type(config["Test"].getboolean("bool1")))
20 print(type(config["Test"]["bool2"]), type(config["Test"].getboolean("bool2")))
21 print(type(config["Test"]["bool3"]), type(config["Test"].getboolean("bool3")))
22
```

```
Section: ['Test']
key : str, value : hello
key : int, value : 123
key : float, value : 3.14
key : bool1, value : True
key : bool2, value : yes
key : bool3, value : 0
<class 'str'> <class 'int'>
<class 'str'> <class 'float'>
<class 'str'> <class 'bool'>
<class 'str'> <class 'bool'>
<class 'str'> <class 'bool'>
```

argparser

- 콘솔창에서 프로그램 실행 시 Setting 정보 저장
- 일반적으로 argparse 모듈 사용
- 머신러닝 등 다양한 코드에서 볼 수 있음

```
python train.py --epochs 50 --batch-size 64 --save-dir weights
```

argparse 사용방법

1. ArgumentParser에 원하는 description을 입력하여 parser 객체 생성
 - usage, default value 등도 지정 가능
2. add_argument() 메소드를 통해 원하는 만큼 인자 종류 추가
 - 이름 정의: 보통 fullname과 약자 하나씩 지정, 이름 앞에 - 붙어있으면 optional
 - 인자 이름에 - 들어갔으면 접근할 때 _로 바꿔줘야 함
 - --help, -h: 인자 사용법에 대한 도움말 출력, 기본으로 내장된 옵션
 - 여러가지 옵션들

type	default	action	dest	nargs	choices	metavar
자료형 지정 (기본값: str)	기본값 지정	Action 종류 지정 (기본값: store)	적용 위치 지정	값 개수 지정	값 범위 지정 (iterable 객체)	이름 재지정

3. parse_args() 메소드로 명령창에서 주어진 인자 파싱
4. args.parameter 형태로 주어진 인자 값 받아서 사용

Example

```
1 import argparse
2
3 parser = argparse.ArgumentParser(description='Argparse Tutorial')
4
5 parser.add_argument('--print-number', '-p', type=int, default = 5,
6 | | | | | help='an integer for printing repeatably')
7
8 args = parser.parse_args()
9
10 for i in range(args.print_number):
11 | print('print number {}'.format(i+1))
12
```

```
[base] macaron:logging bomi$ python arg_test.py --help
usage: arg_test.py [-h] [--print-number PRINT_NUMBER]

Argparse Tutorial

optional arguments:
-h, --help            show this help message and exit
--print-number PRINT_NUMBER, -p PRINT_NUMBER
                      an integer for printing repeatably
[base] macaron:logging bomi$ python arg_test.py --print-number 5
print number 1
print number 2
print number 3
print number 4
print number 5
[base] macaron:logging bomi$ python arg_test.py -p 5
print number 1
print number 2
print number 3
print number 4
print number 5
[base] macaron:logging bomi$ python arg_test.py
print number 1
print number 2
print number 3
print number 4
print number 5
```

Reference

- <https://mingrammer.com/ways-to-manage-the-configuration-in-python/> (파이썬에서 설정값 관리)
- <https://docs.python.org/ko/3/library/configparser.html> (configparser)
- <https://wikidocs.net/13963> (configparser)
- <https://docs.python.org/ko/3/library/argparse.html> (argparse)
- <https://greeksharifa.github.io/references/2019/02/12/argparse-usage/> (argparse)