1. **파이썬 기반의 머신러닝과 생태계 이해**

* **머신러닝의 개념**

**머신러닝이란 ?** 데이터를 기반으로 패턴을 학습하고 결과를 추론하는 알고리즘 기법

**지도 학습이란 ?** 명확한 결정값이 주어진 데이터를 학습하는 것

Ex) 분류 회귀 시각 음성감지/인지

**비지도 학습이란 ?** 결정값이 주어지지 않는 데이터를 학습하는 것

Ex) 군집화(클러스터링)

차원 축소 ( 함축적인 의미 추출도 함)

* **머신러닝의 단점**

**Garbage In Garbage Out** : 데이터에 너무 의존적임

**과적합현상** :

학습데이터에만 충실한 모델을 만들고 실제 데이터를 적용 시 예측성이 떨어지는 현상

* **Numpy ndarray**

: N차원(Dimension) 배열(Array) 객체 / 모두 같은 데이터 타입만 가능

* **Ndarray 형태(shapce)와 차원**

ndarrary.shape (행, 열)

ndarray.ndim 차원 (행) 개수

ndarray.dtype 데이터 타입

ndarray.astype(변경할타입) 데이터 타입 변경

axis 0 : 행 방향 ↓

axis 1 : 열 방향 →

3차원의 경우 axis 0 : 높이 axis 1 : 행 방향 axis 2 : 열 방향  
**- Ndarray 편리하게 생성**

np.arange(10) 0부터 9까지 초기화

np.zeros((3,2), dtype=’int32’) 3행2열 array 0 초기화

np.ones((3,2)) 3행2열 array 1 초기화

* **Ndarray 차원 크기 변경 reshape()**

reshape(**-1**, 5) 이라면 -1은 가변적인 크기라는 의미 뒤에 5는 열 개수는 5로 고정

🡪 즉, 고정된 5개 column에 맞는 row를 자동생성하여 변환

**[ 차원수를 고정하고 싶을 때 ]**

reshape(**-1**, 1) **2차원**으로 변환하되 컬럼 수 1로 고정

reshape(**-1**, ) **1차원**으로 변환

* **Ndarray의 데이터 세트 선택하기 – 인덱싱(Indexing)**

1. **특정 위치의 단일 값 추출**
2. **슬라이싱(Slicing)**
3. **팬시 인덱싱(Fancy Indexing)**

리스트나 ndarray로 인덱스 집합을 지정

[ 1 2 3 4 5 6 7 8 9 ]

Ex) array1[ [2,4,7] ] 인덱스 2,4,7인 값 = [3,5,8]

1. **불린 인덱싱(Boolean Indexing)**

조건 필터링과 검색 동시에 가능

ex) array1[ array1 > 5 ] : array1 값이 5보다 큰 값만 추출

* **Ndarray 배열의 정렬 – sort() 와 argsort()**

np.sort() : 원 행렬 유지 , 정렬된 형렬 반환

ndarray.sort() : 원 행렬이 정렬된 행렬로 변환, 반환 값은 None

* 기본적으로 오름차순, 내림차순은 np.sort()[::-1]

np.sort(A, axis=0) 행 기준 sort // np.sort(A, axis=1) 열 기준 sort

**argsort()**

sort된 원본행렬의 인덱스를 ndarray형으로 반환

* **선형대수 연산**

**np.dot(A,B)** : 행렬 내적

**np.transepose(A)** : 전치 행렬

* **Pandas : 시계열 데이터에 유리**

**DataFrame :** Column Rows **2차원** 데이터 셋

**Series :** Column 값으로만 구성된 **1차원** 데이터 셋 ( column명이 없음 )

**Index :** DataFrame / Series의 고유한 Key값 객체

연산함수를 적용 시, index는 연산에서 제외됨.

DataFrame/Series에 reset\_index() 수행 시 ‘index’ 컬럼명으로 연속형 숫자 데이터할당

|  |  |
| --- | --- |
| **반환 형태** | **설명** |
| list -> DataFrame | df\_list = pd.DataFrame(list, columns=col\_name) |
| ndarray -> DataFrame | df\_array = pd.DataFrame(array, columns=col\_name) |
| dict -> DataFrame | dict = { ‘col1’ : [1,11], ‘col2’ : [2,22] }  **딕셔너리 값은 리스트형태**  df\_dict = pd.DataFrame(dict) |
| DataFrame -> ndarray | DataFrame의 values 속성 |
| DataFrame -> list | DataFrame의 values 속성 + tolist() |
| DataFrame -> dict | DataFrame의 to\_dict() |

* **DataFrame 데이터 삭제**

DataFrame.drop(**labels=None, axis=0,** index=None, columns=None, level=None, inplace=False, **errors=’raise’**)

* axis=0 행 삭제 // axis = 1 컬럼 삭제
* inplace=False는 원본 유지 // inplace=True는 원본 자체가 드롭된 결과

(헷갈리지 않게 drop 명령어에 inplace = False 써주자)

**데이터 셀렉션 및 필터링 – loc, iloc, Boolean indexing**

**loc [] :** 명칭 기반 인덱싱

**iloc[] :** 위치 기반 인덱싱

**Boolean indexing[] :** 조건식을 [] 안에 입력

* **Aggregation 함수**

**sum() , max(), min(), count () :** 집합(Aggregation) 연산 수행

**axis = 0 열** 연산(행방향) **// axis = 1 행** 연산(열방향)

* **DataFrame Group By**

**groupby() :** by 인자에 컬럼명 입력 -> DataFrameGroupBy 객체 반환

* **결손 데이터(Missing Data) 처리하기**

**isna() :** 컬럼값이 NaN인지 True / False

**fillna() :** MissingData를 인자 값으로 대체

* **Pandas apply lambda**

def get\_square(a) : lambda\_square = lambda x : x\*\*2

return a\*\*2

**get\_square(3) lambda\_square(3)**

**apply(lambda x : (식) ) :** 컬럼별로 데이터 가공