

Week 1_예습과제_도하연

1 머신러닝

1.1 개념

- 애플리케이션을 수정하지 않고도 데이터를 기반으로 패턴을 학습하고 결과를 예측하는 알고리즘 기법

1.2 분류

- 지도학습
 - 분류
 - 회귀
 - 텍스트 분석, NLP
- 비지도학습
 - 클러스터링
 - 차원 축소
 - 강화학습

2 넘파이

2.1 넘파이란?

- 머신러닝의 알고리즘은 선형대수와 통계에 기반
- Numerical Python 을 의미하는 NumPy는 선형대수 기반의 프로그램을 지원하는 패키지

2.2 ndarray

2.2.1 개요

- 넘파이의 기반 데이터 타입인 `ndarray` 을 이용해 다차원 배열을 생성하고 사용할 수 있다.
- `shape` 로 `ndarray`의 크기, 행과 열의 수, 배열의 차원 등을 할 수 있다.

```
array1=np.array([1,2,3]) # (3,) 데이터 3개인 1차원
array2=np.array([[1,2,3],[2,3,4]]) #(2, 3) 2개의 열 (row) , 3개의
array3=np.array([[1,2,3]]) # (1,3) # 1개의 row, 3개의 column
```


- 차원
 - `ndarray.ndim` 으로 확인 가능
 - `[]` : 1차원
 - `[[]]` : 2차원

2.2.2 데이터 타입

- 같은 데이터끼리만 연산 가능
- `dtype` 으로 데이터 타입 확인 가능
- `astype` 으로 `ndarray`내 데이터값의 타입 변경

```
int_array.astype('float64') #int 배열을 float로 바꾸기
```

2.2.3 생성 - arange, zeros, ones

 `arange(n)`

0부터 n-1까지의 연속 숫자 값

```
np.arange(10) # [0 1 2 3 4 5 6 7 8 9]
```

🍌 `zeros(튜플의 shape)` `ones(튜플의 shape)`

0 혹은 1로 채운다.

```
zero_array=np.zeros((3,2), dtype='int32')
/* [[0 0]
   [0 0]
   [0 0]] */
```

2.2.4 변경 - reshape

- 특정 차원 및 크기로 변환

```
array1=np.arange(10)
array2=array1.reshape(2,5)
```

- 인자로 -1 을 사용하면 크기에 맞게 자동으로 변환 해준다.

```
array1=np.arange(10) # [0 1 2 3 4 5 6 7 8 9]
```

```
array2=array1.reshape(-1,5) # (2, 5)로 자동 변환
```

- **reshape (-1,1)**
 - 원본 ndarray가 어떤 형태라도 2차원이고, 여러 개의 row를 가지되 반드시 1개의 column을 가진 ndarray로 변환된다.

2.2.5 인덱싱

🍌 단일 값 추출

```
array1d=np.arange(start=1, stop=10)
array2d=array1d.reshape(3,3)
```

```
print('row=0, col=0) :', array2d[0,0])
print('row=0, col=1) :', array2d[0,1])
```

- row는 `axis 0`, column 은 `axis 1` 로 표현하는 것이 정확하다.
- 3차원의 경우 `axis 3` 은 높이를 가리킨다.

🍌 슬라이싱

- 연속한 데이터 추출
- 단일값 추출과 달리 슬라이싱, 팬시 인덱싱, 불린 인덱싱으로 추출된 데이터 세트는 모두 ndarray 타입이다.
- 시작, 종료 인덱싱은 생략 가능

```
array2d[1:3, :]  
/* [[4 5 6]  
   [7 8 9]] */
```

```
array2d[:, :]  
/* [[1 2 3]  
   [4 5 6]  
   [7 8 9]] */
```

```
array2d[0]  
/* [1 2 3] */
```

🍌 펜시 인덱싱

- 인덱스 집합을 지정하면 해당 위치의 인덱스에 해당하는 ndarray 반환
 - `array2d[[0,1],2]` 의 경우 인덱싱이 `(0,2)` `(1,2)` 로 적용되어 `[3.6]`을 반환

🍌 불린 인덱싱

- **조건** 필터링과 검색을 동시에 할 수 있다.
- `array [조건]`

```
array1d # [1 2 3 4 5 6 7 8 9]
array3=array1d[array1d>5] # [6 7 8 9]
```

2.2.6 행렬의 정렬 - sort, argsort

🍌 행렬 정렬 종류

np.sort	ndarray.sort
넘파이에서 호출	행렬 자체에서 호출
원 행렬은 유지. 원 행렬의 정렬된 행렬을 반환	원 행렬 자체를 정렬한 형태로 반환

- 기본적으로 오름차순
- 내림차순은 `np.sort()[::-1]`

🍌 2차원 행렬 정렬

```
array2d=np.array([[8,12],[7,1]])
sort_array2d_axis0=np.sort(array2d, axis=0)
print('로우 방향으로 정렬:\n', sort_array2d_axis0)

sort_array2d_axis1=np.sort(array2d, axis=1)
print('칼럼 방향으로 정렬:\n', sort_array2d_axis1)
```

🍌 정렬된 행렬의 인덱스 반환 `argsort`

```
org_array=np.array([3,1,9,5])
sort_indices=np.argsort(org_array) # [1 0 3 2]
```

2.2.7 선형대수 연산

🍌 행렬 곱 `np.dot(행렬, 행렬)`

🍌 전치 행렬 `np.transpose(행렬)`

3 판다스

3.1 판다스란?

- 파이썬에서 데이터 처리를 위한 라이브러리
- 리스트, 넘파이 등의 내부 데이터 뿐만 아니라 CSV 파일을 쉽게 DataFrame 으로 변경 가능
- 용어
 - DataFrame: 여러 개의 행과 열로 이뤄진 2차원 데이터를 담은 데이터 구조체
 - Index: 개별 데이터를 고유하게 식별하는 Key값
 - Series: 칼럼이 하나뿐인 데이터 구조체

3.2 판다스 API

- `read_csv`
 - CSV 파일의 포맷을 DataFrame 으로 변환.
 - 칼럼 구분 문자 → 점 (.)
 - 디폴트 필드 구분 문자 → 콤마 (,)
 - 다른 구분 문자 기반의 파일 포맷도 변환 가능

```
read_csv('파일명', sep='/t')
```

- `read_table`
 - 필드 구분 문자 → 탭 (\t)

- `read_fwf`
 - 고정 길이 기반의 칼럼 포맷을 DataFrame으로 로딩하기 위한 API
- `head(n)`
 - 맨 앞 n개의 row 반환
- `shape`
 - 행과 열의 크기
- `info`
 - 총 데이터 건수, 데이터 타입, Null 건수
- `describe`
 - 칼럼별 숫자형 데이터 값의 n-percentile 분포도, 평균값, 최댓값, 최솟값
 - object 타입은 포함하지 않는다
- `DataFrame['칼럼명']`
 - series 형태로 특정 column 데이터 세트가 반환된다.
- `value_counts()`
 - column값의 유형과 건수를 확인할 수 있다.

```
titanic_df['Pclass'].value_counts()
```

- Series(→ Index와 단 하나의 칼럼으로 구성된 데이터 세트)에서만 사용 가능
- DataFrame에서는 사용 불가능

3.3 DataFrame 과 ndarray 상호 변환

- DataFrame은 리스트, 넘파이, ndarray와 다르게 칼럼명 필요

3.3.1 ndarray, 리스트, 딕셔너리 → DataFrame

🍊 1차원

```
col_name1=['칼럼 이름']
list=[1,2,3]
array1=np.array(list1)

#넘파이 ndarray를 이용해 DataFrame생성
df_array1=pd.DataFrame(array1, columns=col_name1)
```

🍊 2차원

```
# 3개의 칼럼명이 필요함
col_name2 = ['col1', 'col2', 'col3']

# 리스트
list2 = [[1, 2, 3],
          [11, 12, 13]]

# DataFrame으로 변환
df_list2 = pd.DataFrame(list2, columns=col_name2)
```

🍊 딕셔너리

- key는 칼럼명, value 는 각 칼럼 데이터가 된다.

```
dict = {'col1':[1, 11], 'col2':[2, 22], 'col3':[3, 33]}
df_dict = pd.DataFrame(dict)
```


3.3.2 DataFrame → ndarray, 리스트, 딕셔너리

🍌 ndarray

- `values` 이용

```
df_dict.values
```

🍌 리스트 / 딕셔너리

- `tolist` / `to_dict` 이용

```
df_dict.values.tolist() # 리스트로  
df_dict.to_dict('list') # 딕셔너리가 리스트형으로 반환
```

3.4 칼럼 데이터 생성, 수정, 삭제

3.4.1 생성, 수정

- `DataFrame['칼럼명'] = 일괄적으로 할당되는 값 혹은 계산 식`

```
titanic_df['Age_0']=0  
titanic_df['Family_No'] = titanic_df['SibSp'] + titanic_df['Par...
```

3.4.2 삭제

`drop`

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

- 🍌 `axis`
 - `labels`에 원하는 칼럼명 & `axis=1` 입력하면 지정된 칼럼 드롭

```
titanic_drop_df = titanic_df.drop('Age_0', axis=1 )
```

- axis=0을 입력하면 로우 축 방향으로 드롭
- 🍌 inplace
 - False이면 자기 자신의 DataFrame의 데이터는 삭제하지 않으며, 삭제된 결과는 별도의 새로운 DataFrame으로 반환.
 - True이면 원본 DataFrame에 드롭된 결과 적용된다.

3.5 Index 객체

- 레코드를 고유하게 식별하는 객체
- Index는 오직 식별용으로만 사용되기 때문에 Series 객체에 연산 함수 적용할 때 Index는 연산에서 제외된다.

3.5.1 Index 객체 추출

- ndarray와 유사하게 단일 값 반환 및 슬라이싱 가능

```
indexes = titanic_df.index  
indexes[:5].values # [0 1 2 3 4]
```

3.5.2 reset_index

- 연속 숫자형으로 새롭게 인덱스를 할당
- 기존 인덱스는 index 라는 새로운 칼럼명으로 추가된다.
 - drop=True 설정하면 새로 추가되지 않고 삭제된다.

3.6 데이터 셀렉션 및 필터링

3.6.1 DataFrame의 [] 연산자

- 넘파이나 Series의 []와 다르다!

- 칼럼명만 지정할 수 있다.

```
titanic_df[ ['Survived', 'Pclass'] ]
titanic_df[0] #오류
```

- 슬라이싱 및 불린 인덱싱은 가능

```
titanic_df[0:2]
titanic_df[ titanic_df['Pclass'] == 3]
```

3.6.2 명칭 기반 인덱싱 / 위치 기반 인덱싱

🍊 명칭 기반 인덱싱

- 칼럼 명으로 열 위치 지정

🍊 위치 기반 인덱싱

- 행, 열 값으로 정수

3.6.4 iloc[]

- 위치 기반 인덱싱
 - 행과 열 값으로 정수형 또는 슬라이싱, 랜시 리스트 값 입력

```
data_df.iloc[0, 0] # Chulmin
data_df.iloc[0, 'Name'] # 오류
```

3.6.5 loc[]

- 명칭 기반 인덱싱
 - 행 → DataFrame index 값
 - 열 → 칼럼 명

```
data_df.loc['one', 'Name'] # Chulmin
```

3.6.6 불린 인덱싱

`DataFrame[조건]`

```
titanic_df[ (titanic_df['Age'] > 60) & (titanic_df['Pclass']==1)
```

3.7 정렬

3.7.1 sort_values

- `by`

- 특정 칼럼 입력하면 해당 칼럼으로 정렬

```
titanic_df.sort_values(by=['Name'])
```

- `ascending`

- `ascending=True` : 오름차순
- `False` : 내림차순

- `inplace`

- `inplace=False` : 호출한 DataFrame은 그대로 유지하며 정렬도니 DataFrame을 결과로 반환

3.7.2 Aggregation 함수

- `min, max, sum, count` 함수

3.7.3 groupby

- 입력 파라미터 `by`에 칼럼을 입력하면 대상 칼럼으로 `groupby` 된다.
- SQL과 다르게 DataFrame에 `groupby`를 호출해 반환된 결과에 `agggregation` 함수를 호출하면 `groupby` 대상 칼럼을 제외한 모든 칼럼에 해당 `aggregation` 함수를 적용

- 객체에 해당 칼럼을 필터링한 뒤 aggregation 함수 적용해야 한다.

```
titanic_df.groupby('Pclass')[['PassengerId', 'Survived']].count()
```

3.8 결손 데이터 처리

머신러닝 알고리즘은 NaN 값 처리하지 않는다.

3.8.1 isna - 결손 데이터 여부 확인

```
titanic_df.isna().head(3)
```

3.8.2fillna - 결손 데이터 대체

```
# age 칼럼의 NaN 값은 평균 나이로 대체
titanic_df['Age'] = titanic_df['Age'].fillna(titanic_df['Age'].r

# Embarked 칼럼의 NaN 값은 'S' 로 대체
titanic_df['Embarked'] = titanic_df['Embarked'].fillna('S')
```

3.9 apply lambda - 데이터 가공

3.9.1 lambda 식

3.9.2 DataFrame의 lambda 식

```
titanic_df['Name_len']= titanic_df['Name'].apply(lambda x : len(x))
```

3.9.2 if else

- if 절의 경우 if 식보다 반환 값을 먼저 기술해야 한다.