1. 파이썬 기반의 머신러닝과 생태계 이해

1. 머신러닝의 개념

머신러닝: 애플리케이션을 수정하지 않고도 데이터를 기반으로 패턴을 학습하고 결과를 예측하는 알고리즘 기법

머신러닝의 분류 지도학습(Supervised Learning)

- 분류(Classification)
- 회귀(Regression)
- 추천시스템
- 시각/음성감지/인지
- 텍스트 분석, NLP

비지도학습(Un—supervised Learning)

- 클러스터링
- 차원축소
- 강화학습(Reinforcement Learning)

2. 파이썬 머신러닝 생태계를 구성하는 주요 패키지

- 머신러닝 패키지: 사이킷런 등
- 행렬/선형대수/통계 패키지: 넘파이 등
- 데이터 핸들링: 넘파이, 판다스 등
- 시각화: 맷플롯립, 시본 등

3. 넘파이

넘파이(NumPy;Numerical Python): 파이썬에서 선형대수 기반의 프로그램을 쉽게 만들 수 있도록 지원하는 대표적인 패키지 넘파이의 기반 데이터 타입: ndarray

- np.array(): ndarray로 변환을 원하는 객체를 인자로 입력하면 ndarray를 반환
- ndarray.shape: ndarray의 차원과 크기를 튜플(tuple) 형태로 나타냄 ex) [1,2,3]인 arrayl의 shape는 (3,) -> 1차원 array로 3개의 데이터를 가지고 있음 [[1,2,3], [2,3,4]]인 array의 shape는 (2,3) -> 2차원 array로 2개의 row와 3개의 columnd으로 구성되어 6개의 데이터 가지고 있음
- array(): 함수의 인자 -> 주로 리스트 객체. []는 1차원, [[]]는 2차원
- ndarray.ndim: array의 차원 나타냄

ndarray의 데이터타입 ndarray내의 데이터값은 숫자 값, 문자열 값, 불 값 등이 모두 가능 리스트: 서로 다른 데이터 타입을 가질 수 있음 ndarray: 연산의 특성상 같은 데이터 타입만 가질 수 있음. 만약 다른 데이터 유형이 섞여 있는 리스트를 ndarray로 변경하면 데이터 크기가 더 큰 데이터 타입으로 형 변환을 일괄 적용 - ndarray.dtype: dtype 속성 나타냄 -astype(): ndarray 내 데이터값의 타입 변경. 메모리 절약(float을 int로 변경 등)

ndarray를 편리하게 생성하기 - arange, zeros, ones

- arange(): array를 range()로 표현

- zeros(): 함수 인자로 튜플 형태의 shape 값을 입력하면 모든 값을 0으로 채운 해당 shape를 가진 ndarray를 반환
- ones(): 함수 인자로 튜플 형태의 shape 값을 입력하면 모든 값을 1로 채운 해당 shape를 가진 ndarray를 반환 함수 인자로 dtype을 정해주지 않으면 default 로 float64형의 데이터로 ndarray를 채움

ndarray 의 차원과 크기를 변경하는 reshape()

- reshape(): ndarray를 특정 차원 및 크기로 변환 -1값을 인자로 적용하면 자동으로 새롭게 생성해 변환 ex) ndarray가 1차원이며 10개의 데이터를 가지고 있을 때, ndarray.reshape(-1,5)를 실행하면 row는 자동으로 2. reshape(-1,1): 원본 ndarray가 어떤 형태라도 2차원이고, 여러 개의 로우를 가지되 반드시 1개의 칼럼을 가진 ndarray로 변환됨을 보장

- tolist(): ndarray를 리스트 자료형으로 변

넘파이의 ndarray의 데이터 세트 선택하기 - 인덱싱(Indexing)

1. 특정한 데이터만 추출: 원하는 위치의 인덱스 값을 지정하면 해당 위치의 데이터가 반환 array[0], array[-1], array[-2], array[row,col] 단일 인덱스를 이용해 ndarray 내의 데이터값도 간단히 수정 가능

- 다차원 array에서 axis0=row, axis1=column, axis2....

2. 슬라이싱(Slicing): 슬라이싱은 연속된 인덱스상의 ndarray를 추출하는 방식 ' : ' 기호 사이에 시작 인덱스와 종료 인덱스를 표시하면 시작 인덱스에서 종료 인덱스-1 위치에 있는 데이터의 ndarray를 반환 ex) 1 : 5라고 하면 시작 인덱스 1과 종료 인덱스 4까지에 해당하는 ndarray를 반환 1)' : ' 기호 앞에 시작 인덱스를 생략하면 자동으로 맨 처음 인덱스인 0으로 간주합니다.

2. ':' 기호 뒤에 종료 인덱스를 생략하면 자동으로 맨 마지막 인덱스로 간주합니다.
3. ' : ' 기호 앞/뒤에 시작/종료 인덱스를 생략하면 자동으로 맨 처음/맨 마지막 인덱스로 간주합니다.

2차원 ndarray에서는 콤마(,)로 로우와 칼럼 인덱스를 지칭 ex) array(0:2,0:2), array(: , :)

2차원 ndarray에서 뒤에 오는 인덱스를 없애면 1차원 ndarray를 반환. 3차원에서 뒤에 오는 인덱스를 없애면 2차원 ndarray를 반

3. 팬시 인덱싱(Fancy Indexing) : 일정한 인덱싱 집합을 리스트 또는 ndarray 형태로 지정해 해당 위치에 있는 데이터의 ndarray를 반환 리스트나 ndarray로 인덱스 집합을 지정하면 해당 위치의 인덱스에 해당하는 ndarray를 반환하는 인덱싱 방식 ex)array[[0,1],2] -> row(axis0)축에 인덱스0,1, column(axis1)축에 인덱스2

4. 불린 인덱싱(Boolean Indexing) : 특정 조건에 해당하는지 여부인 True/False 값 인덱싱 집합을 기반으로 True 에 해당하는 인덱스 위치에 있는 데이터의 ndarray를 반환 ex) array[array>5] Step 1 : array>5와 같이 ndarray의 필터링 조건을 [] 안에 기재 Step 2 : False 값은 무시하고 True 값에 해당하는 인덱스값만 저장 (유의해야 할 사항은 True값 자체인 1을 저장하는 것 이 아니라 True값을 가진 인덱스를 저장한다는 것입니다) Step 3 : 저장된 인덱스 데이터 세트로 ndarray 조회

행렬의 정렬 — sort()와 argsort()

- np.sort(): 원 행렬은 그대로 유지한 채 원 행렬의 정렬된 행렬을 반환
- ndarray.sort()는 원 행렬 자체를 정렬한 형태로 변환하며 반환 값은 None

내림차순으로 정렬하기 위해서는 [::-1]을 적용 ex) np.sort( )[::-1]과 같이 사용

행렬이 2차원 이상일 경우에 axis 축 값 설정을 통해 로우 방향, 또는 칼럼 방향으로 정렬을 수행 ex) np.sort(array, axis=0)

- np.argsort(): 정렬 행렬의 원본 행렬 인덱스를 ndarray 형으로 반환 오름차순이 아닌 내림차순으로 정렬 시 np.argsort( )[::-1]

선형대수 연산 - 행렬 내적과 전치 행렬 구하기

- np.dot(): 행렬 내적(행렬 곱)
- np.transpose(): 전치행렬 구하기

4. 데이터 핸들링 - 판다스

판다스의 핵심 객체: DataFrame(여러 개의 행과 열로 이뤄진 2차원 데이터를 담는 데이터 구조체)

판다스 시작 - 파일을 DataFrame으로 로딩, 기본 API -read_csv(filepath_or_buffer, sep=', ',...): 다양한 포맷으로 된 파일을 DataFrame으로 로딩 -dataframe.head( ): DataFrame의 맨 앞에 있는 N개의 로우를 반환 -dataframe.shape(): 행과 열을 튜플 형태로 반환 -dataframe.info( ): 통해서 총 데이터 건수와 데이터 타입, Null 건수를 알 수 있음 -dataframe.describe(): 칼럼별 숫자형 데이터값의 n-percentile 분포도, 평균값, 최댓값, 최솟값을 나타냄. 오직 숫자형(hit, float 등) 칼럼의 분 포도만 조사하며 자동으로 object 타입의 칼럼은 출력에서 제외

- dataframe[].value_counts(): 해당 칼럼값의 유형과 건수를 확인 데이터 타입은 series(Index와 단하나의 칼럼으로 구성된 데이터 세트). Null값을 포함하려면 dropna 인자값을 FALSE로 입력. value_counts(dropna=FALSE)

DataFrame과 리스트, 딕셔너리, 넘파이 ndarray 상호 변환 DataFrame은 리스트와 넘파이 ndarray와 다르게 칼럼명을 가지고 있음 DataFrame은 기본적으로 행과 열을 가지는 2차원 데이터. 2차원 이하의 데이터들만 dataframe으로 변환될 수 있음

- 2행 3열 리스트, 넘파이 ndarray를 DataFrame으로 변 col_name=['','',''] list = [[,],[,,]] pd.DataFrame(list, columns=col_name)

col_name=['','',''] array = np.array(list) pd.DataFrame(array, columns=col_name)

- 딕셔너리를 DataFrame으로 변환 dict = {'col1':[,], 'col2':[,], 'col3':[,]} df_dict = pd.DataFrame(dict)
- DataFrame을 넘파이 ndarray로 변환하기 Dataframe.values
- DataFrame을 리스트로 변환 Dataframe.values.tolist()
- DataFrame을 딕셔너리로 변환 Dataframe.to_dict('list')

DataFrame의 칼럼 데이터 세트 생성과 수정 DataFrame[] 내에 칼럼명을 입력하고 값을 할당

DataFrame 데이터 삭제 DataFrame.drop(labels=None, axis=0, index=None, columns=None, level=None, inplace=False, errors='raise') 칼럼 드롭 시에는 labels=원하는 칼럼명 리스트, axis=1 로우 드롭 시에는 labels=인덱스, axis=0 자신의 원본 DataFrame의 데이터를 아예 삭제할 시에는 inplace = TRUE

- axis： DataFrame의 로우를 삭제할 때는 axis=0, 칼럼을 삭제할 때는 axis=I으로 설정.
- 원본 DataFrame은 유지하고 드롭된 DataFrame을 새롭게 객체 변수로 받고 싶다면 inplace=False로 설정 (디폴트 값이 False 임). ex) titanic_drop_df = titanic_df.drop(' Age_0', axis=1, inplace=False)

- 원본 DataFrame에 드롭된 결과를 적용할 경우에는 inplace=True를 적용. ex) titanic_df.drop('Age_0', axis=l, inplace=True)
- 원본 DataFrame에서 드롭된 DataFrame을 다시 원본 DataFrame 객체 변수로 할당하면 원본 DataFrame에서 드롭된 결과를 적용할 경우와 같음 (단, 기존 원본 DataFrame 객체 변수는 메모리에서 추후 제거됨). ex) titanic_df = titanic_df.drop('Age_0', axis=1, inplace=False)

Index 객체

- Index는 오직 식별용으로만 사용.
- Series 객체는 Index 객체를 포함하지만 Series 객체에 연산 함수를 적용할 때 Index는 연산에서 제외됨
- DataFrame 및 Series에 reset_index() 메서드를 수행하면 새롭게 인덱스를 연속 숫자 형으로 할당하며 기존 인덱스는 'index'라는 새로운 칼럼 명으로 추가 Series에 reset_index()를 적용하면 새롭게 연속 숫자형 인덱스가 만들어지고 기존 인덱스는 'index' 칼럼명으로 추가되면서 DataFrame으로 변환됨
- reset_index( )의 parameter 중 drop=True로 설정하면 기존 인덱스는 새로운 칼럼으로 추가되지 않고 삭제(drop)됨

데이터 셀렉션 및 필터링 < DataFrame[] 연산자> DataFrame 바로 뒤에 있는 '[ ]' 안에 들어갈 수 있는 것: 칼럼명 문자(또는 칼럼명의 리스트 객체), 또는 인덱스로 변환 가능한 표현식(슬라이싱, 불린 인덱싱)
ex)titanic_df['Pclass']가능, titanic_df[0:2] 가능, titanic_df[0]은 불가능

- DataFrame 바로 뒤의 [] 연산자는 넘파이의 [ ]나 Series의 [ ]와 다릅니다.
- DataFrame 바로 뒤의 [] 내 입력값은 칼럼명(또는 칼럼의 리스트)을 지정해 칼럼 지정 연산에 사용하거나 불린 인덱스용도로만 사용해야 합니다.
- DataFrame[0:2]와 같은 슬라이싱 연산으로 데이터를 추출하는 방법은 사용하지 않는 게 좋습니다.

< DataFrame iloc [] 연산자>

- iloc[ ]는 위치(Location) 기반 인덱싱 방식으로 동작
- iloc[ ]는 위치 기반 인덱싱만 허용하기 때문에 행과 열의 좌표 위치에 해당하는 값으로 정숫값 또는 정수형의 슬라이싱, 팬시 리스트 값을 입력해줘야 함
- iloc[ ]에 DataFrame의 인덱스 값이나 칼럼명을 입력하면 오류를 발생 ex) iloc[:, -1]-> 맨 마지막 칼럼의 값, 즉 타깃 값을 가져옴 ex) iloc[:, :-1]-> 처음부터 맨 마지막 칼럼을 제외한 모든 칼럼의 값, 즉 피처값들을 가져오게 됨

< DataFrame loc[] 연산자>

- loc[ ]는 명칭(Label)기반 인덱싱 방식으로 동작
- loc[인덱스값, 칼럼명]
- !!!주의!!! loc[ ]에 슬라이싱 기호를 적용하면 종료 값-1이 아니라 종료 값까지 포함

<불린 인덱싱>

- [], loc[ ]에서 공통으로 지원
- iloc[] 는 정수형 값이 아닌 불린 값에 대해서는 지원하지 않기 때문에 불린 인덱싱이 지원 안 됨

1. and 조건일 때는 &
2. or 조건일 때는 |
3. Not 조건일 때는 ~ ex) titanic_df[titanic_df['Age'] > 60]

정렬, Aggregation 함수, GroupBy 적용 < sort_values() >

- DataFrame과 Series의 정렬
- by=['특정 칼럼'], ascending=TRUE/FALSE, inplace=FALSE/TRUE
- 여러 칼럼으로 정렬하면 앞 칼럼부터 정렬됨

< Aggregation 함수 적용>

- Aggregation 함수: min( ), max( ), sum( ), count( ) 등
- 특정 칼럼에 aggregation 함수를 적용하기 위해서는 DataFrame에 대상 칼럼들만 추출해 aggregation을 적용

< groupby() 적용 >

- by에 칼럼 입력하면 대상 칼럼으로 groupby 됨
- DataFrame에 groupby()를 호출하면 DataFrameGroupBy라는 또 다른 형태의 DataFrame을 반환
- DataFrame에 groupby()를 호출해 반환된 결과에 aggregation 함수를 호출하면 groupby() 대상 칼럼을 제외한 모든 칼럼에 해당 aggregation 함수를 적용
- DataFrame의 groupby( )에 특정 칼럼만 aggregation 함수를 적용하려면 grouby( )로 반환된 DataFrameGroupBy 객체에 해당 칼럼을 필터링한 뒤 aggregation 함수를 적용
- DataFrame의 groupby( )에 여러 개의 칼럼이 서로 다른 aggregation 함수를 호출하려면 agg()내에 딕셔너리 형태로 칼럼들과 aggregation 함수를 입력 ex) agg_format={'Age' : 'max', 'SibSp':'sum', 'Fare':'mean'} titanic_df.groupby('Pclass').agg(agg_format)

결손 데이터 처리하기 결손 데이터: 칼럼에 값이 없는, NULL인 경우를 의미. 넘파이의 NaN로 표시.

- isna(): NaN 여부를 확인하는 API
- DataFrame.isna().sum(): 결손 데이터의 개수
- fillna(): NaN 값을 다른 값으로 대체하는 API. inplace=True를 설정해야 실제 원본 데이터 세트 값이 변경됨. ex) titanic_df['Cabin'].fillna('C000',inplace=TRUE)

apply lambda 식으로 데이터 가공 lambda 식: 함수의 선언과 함수 내의 처리를 한 줄의 식으로 쉽게 변환하는 식. ex) lambda x : x ** 2 -> ':'의 왼쪽에 있는 x는 입력 인자를 가리키며, 오른쪽은 입력 인자의 계산식(반환 값)

lambda 식을 이용할 때 여러 개의 값을 입력 인자로 사용해야 할 경우 : map() ex) a=[1, 2, 3] squares = map(lambda x : x**2, a) list(squares)

DataFrame의 lambda 식: DataFrame['칼럼'].apply(lambda x:) if else절 이용할 때, if 절의 경우 if 식보다 반환 값을 먼저 기술: DataFrame['칼럼'].apply(lambda x: '반환값' if 조건 else '반환값') else if는 지원하지 않음. else if 이용하려면: DataFrame['칼럼'].apply(lambda x: '반환값' if 조건 else ('반환값' if 조건 else '반환값'))