[파머완] 01 파이썬 기반의 머신러닝과 생태계 이해

진규빈 · 방금 전

파이썬 머신러닝 완벽 가이드

목록 보기 1/1

(1) 머신러닝의 개념

> - 애플리케이션을 수정하지 않고도 데이터를 기반으로 패턴을 학습하고 결과를 예측하는 알고리즘 기법
> - 데이터 기반으로 통계적 신뢰도 강화하고 예측 오류 최소화하기 위한 다양한 수학적 기법 적용해 데이터 내의 패턴 스스로 인지하고 신뢰도 있는 예측 결과 도출

- 분류
  <**지도학습**>
  분류, 회귀, 추천 시스템, 시각/음성 감지/인지, 텍스트 분석, NLP
  <**비지도학습**>
  클러스터링, 차원 축소, **강화학습**

- 단점
  - 데이터에 매우 의존적
  - 가비지 인(Garbage In) & 가비지 아웃(Garbage out)
  - 데이터 품질이 좋지 않으면 머신러닝 수행 결과도 좋을 수 없음

∴ 최적의 머신러닝 알고리즘과 모델 파라미터 구축하는 능력도 중요하지만 데이터 이해하고 효율적으로 가공, 처리, 추출해 최적의 데이터 기반으로 알고리즘 구동할 수 있도록 준비하는 능력이 더 중요!

- 파이썬과 R 기반의 머신러닝 비교
  <파이썬>
  - 직관적인 문법
  - 객체지향과 함수형 프로그래밍 모두를 포함하는 유연한 프로그램 아키텍쳐
  - 다양한 라이브러리
  - 머신러닝 개발자에게 더 권장!
  <R>
  - 통계 전용 프로그램 언어
  - 다양하고 많은 통계 패키지 보유

(2) 파이썬 머신러닝 생태계를 구성하는 주요 패키지

- 머신러닝 패키지
  - 사이킷런(Scikit-Learn): 대표적 패키지

- 행렬/선형대수/통계 패키지
  - 머신러닝의 이론적 백그라운드가 선형대수와 통계로 이루어져 있음
  - 넘파이(NumPy): 대표적 패키지
  - 사이파이(SciPy): 자연과학과 통계 위한 다양한 패키지 가짐

- 데이터 핸들링
  - 판다스: 대표적 데이터 처리 패키지, 2차원 데이터 처리에 특화

- 시각화
  - 맷플롯립(MatPlotlib): 대표적 시각화 패키지, 너무 세분화된 API로 익히기 번거로우며 디자인적으로 투박한 면 존재, 효율성 떨어지고 불편함 늘어날 수 있음
  - 시본(Seaborn): 맷플롯립 보완하기 위해 이를 기반으로 출시된 시각화 패키지, 판다스와 연동 더 쉬움, 더 함축적 API, 분석 위한 다양한 유형의 그래프 및 차트 제공

- 이외 여러 서드파티 라이브러리 존재

- 아이파이썬(IPython, Interactive Python): 대화형 파이썬 툴로, 프로그래밍과 이에 대한 설명적 요소 결합

- 주피터 노트북(Jupyter Notebook): 대표적 아이파이썬 지원 툴

(3) 넘파이

> - Numpy, Numerical Python
> - 파이썬에서 선형대수 기반의 프로그램 쉽게 만들 수 있도록 지원하는 대표적 패키지

- 루프 사용하지 않고 대량 데이터의 배열 연산 가능하게 하므로 빠른 배열 연산 속도 보장
- C/C++과 같은 저수준 언어 기반의 호환 API 제공
  - 기존 C/C++ 기반의 타 프로그램과 데이터 주고받거나 API 호출해 쉽게 통합할 수 있는 기능 제공
  - 수행 성능이 매우 중요한 부분은 C/C++ 기반의 코드로 작성하고 이를 넘파이에서 호출하는 방식으로 쉽게 통합 가능
- 배열 기반의 연산은 물론이고 다양한 데이터 핸들링 기능 제공
- BUT 편의성과 다양한 API 지원 측면에서 아쉬운 부분 많음! 2차원 형태의 행과 열로 이루어진 데이터에 대한 다양한 가공과 변환, 여러 통계용 함수의 적용 등의 측면에서 판다스가 더 편리함
- 그럼에도 넘파이를 이해하는 것은 파이썬 기반의 머신러닝에서 매우 중요~ 많은 머신러닝 알고리즘이 넘파이 기반으로 작성돼 있으며, 이들 알고리즘의 입출력 데이터를 넘파이 배열 타입으로 사용하기 때문

*** ndarray**

- 넘파이의 기반 데이터 타입
- 다차원 배열 쉽게 생성하고 다양한 연산 수행 가능
- 넘파이 `array()` 함수는 파이썬의 리스트와 같은 다양한 인자 입력받아 ndarray로 변환하는 기능 수행
  - 생성된 ndarray 배열의 shape 변수는 ndarray의 크기, 즉 행과 열의 수를 튜플 형태로 갖고 있으며 이를 통해 ndarray 배열의 차원까지 알 수 있음
  - `np.array()` => ndarray로 변환 원하는 객체를 인자로 입력하면 ndarray를 반환! `ndarray.shape` 는 ndarray의 차원과 크기를 튜플 형태로 나타내줌
- ndarray 내의 데이터값은 숫자 값, 문자열 값, 불 값 등 모두 가능
  - 숫자형의 경우 int형(8bit, 16bit, 32bit), unsigned int형(8bit, 16bit, 32bit), float형(16bit, 32bit, 64bit, 128bit), complex 타입 제공

- dtype 속성으로 데이터 타입 확인 가능
- 만약 다른 데이터 유형 섞여 있는 리스트를 ndarray로 변경하면 데이터 크기가 더 큰 데이터 타입으로 형 변환 일괄 적용
- `astype()` 메서드 이용해 데이터값 타입 변경 가능 => 인자로 원하는 타입을 문자열로 지정하면 됨 (메모리 더 절약해야 할 때 보통 이용)

- `arange()`, `zeros()`, `ones()`
  - 특정 크기와 차원 가진 ndarray를 연속값이나 0 또는 1로 초기화해 쉽게 생성해야 할 필요 있는 경우 이용
  - 주로 테스트용으로 데이터 만들거나 대규모의 데이터 일괄적으로 초기화해야 할 경우에 사용
  - `arange()` => 파이썬 표준 함수 `range()` 와 유사한 기능! array를 `range()` 로 표현하는 것으로, 0부터 함수 인자 값 -1까지의 값을 순차적으로 ndarray의 데이터값으로 변환
  - `zeros()` => 함수 인자로 튜플 형태의 shape 값 입력하면 모든 값을 0으로 채운 해당 shape 가진 ndarray 반환
  - `ones()` => 함수 인자로 튜플 형태의 shape 값 입력하면 모든 값을 1로 채운 해당 shape 가진 ndarray 반환
  - 함수 인자로 dtype 정해주지 않으면 default로 float64 형의 데이터로 ndarray 채움

- `reshape()`
  - ndarray를 특정 차원 및 크기로 변환
  - 변환 원하는 크기를 함수 인자로 부여
  - 지정된 사이즈로 변경 불가능하면 오류 발생
  - 인자로 -1을 적용하면 원래 ndarray와 호환되는 새로운 shape으로 변환 (효율적 실전 활용)

- 인덱싱(Indexing)
  1) 특정한 데이터만 추출: 원하는 위치의 인덱스 값 지정하면 해당 위치의 데이터 반환
  2) 슬라이싱(Slicing): 연속된 인덱스상의 ndarray 추출
  - ':' 기호 사이에 시작 인덱스와 종료 인덱스 표시하면 시작 인덱스에서 종료 인덱스-1 위치에 있는 데이터의 ndarray 반환
  - ':' 기호 앞에 시작 인덱스 생략 시 자동으로 맨 처음 인덱스인 0으로 간주
  - ':' 기호 뒤에 종료 인덱스 생략 시 자동으로 맨 마지막 인덱스로 간주
  - ':' 기호 앞/뒤에 시작/종료 인덱스 생략 시 자동으로 맨 처음/맨 마지막 인덱스로 간주
  3) 팬시 인덱싱(Fancy Indexing): 일정한 인덱스 집합을 리스트 또는 ndarray 형태로 지정해 해당 위치에 있는 데이터의 ndarray 반환
  4) 불린 인덱싱(Boolean Indexing): 특정 조건에 해당하는지 여부인 True/False 값 인덱싱

- for loop / if else 문보다 훨씬 간단하게 구현 가능
  - 내부적으로 여러 단계 거쳐 동작하지만 코드 자체는 단순히 [] 내에 원하는 필터링 조건만 넣으면 해당 조건 만족하는 ndarray 데이터 세트 반환하기 때문에 사용자는 내부 로직에 크게 신경 안 쓰고 쉽게 코딩 가능

- `sort()`, `argsort()`
  - `np.sort()`, `ndarray.sort()` : 넘파이에서 행렬 정렬하는 대표적 방법
    - `np.sort()` 는 원 행렬 그대로 유지한 채 원 행렬의 정렬된 행렬 반환, `ndarray.sort()` 는 원 행렬 자체를 정렬한 형태로 변환하며 반환 값은 None
    - `np.argsort()` : 정렬된 행렬의 원본 행렬 인덱스를 ndarray 형으로 반환
    - 오름차운 아닌 내림차순으로 정렬 시 [::-1] 적용

- 선형대수 연산
  - 행렬 내적(행렬 곱): 두 행렬 A와 B의 내적은 `np.dot()` 이용해 계산 가능
    - 전치 행렬: `transpose()` 이용

(4) 데이터 핸들링 - 판다스

- 파이썬에서 가장 인기 있는 데이터 처리 라이브러리
- 행과 열로 이루어진 2차원 데이터를 효율적으로 가공 및 처리할 수 있는 다양한 기능 제공
- 저수준 API가 대부분인 넘파이보다 훨씬 유연하고 편리하게 데이터 핸들링 가능하게 해줌

*** DataFrame**

> - 판다스의 핵심 개체로, 여러 개의 행과 열로 이루어진 2차원 데이터를 담는 데이터 구조체
> - 파이썬의 리스트, 컬렉션, 넘파이 등의 내부 데이터뿐만 아니라 CSV 등의 파일을 쉽게 DataFrame으로 변경해 데이터의 가공/분석 편리하게 수행할 수 있게 만들어줌

- Index & Series
  - Index: RDBMS의 PK처럼 개별 데이터를 고유하게 식별하는 Key 값
  - Series와 DataFrame 모두 Index를 key 값으로 갖고 있음
  - Series는 칼럼이 하나뿐인 / DataFrame은 칼럼이 여러 개인 데이터 구조체
  - DataFrame은 여러 개의 Series로 이루어짐

- pd.read_csv() 호출 시 파일명 인자로 들어온 파일을 로딩해 DataFrame 객체로 반환

- read_table() : read_csv() 와 같은 기능 BUT 필드 구분 문자가 콤마 말고 탭
- read_fwf() : 고정 길이(Fixed Width) 기반의 칼럼 포맷을 DataFrame으로 로딩하기 위한 API
- DataFrame.head() : DataFrame의 맨 앞에 있는 N개의 로우 반환
- DataFrame.shape() : DataFrame의 행과 열을 튜플 형태로 반환
- info() : 총 데이터 건수, 데이터 타입, Null 건수
- describe() : 칼럼별 숫자형 데이터값의 n-percentile 분포도, 평균값, 최댓값, 최솟값 (오직 숫자형 칼럼의 분포도만 조사, object 타입 칼럼은 자동으로 출력에서 제외)
- value_counts() : DataFrame의 [] 연산자 내부에 칼럼명 입력하면 Series 형태로 특정 칼럼 세트가 반환되고 반환된 객체에 이 메서드 호출 시 지정된 칼럼의 데이터값 건수 반환
  - 데이터 분포도 확인하는 데 매우 유용한 함수
  - 칼럼 값별 데이터 건수 반환하므로 고유 칼럼 값을 식별자로 사용 가능
  - Null 값 무시하고 결과값 내놓기 쉽다는 점 유의 (Null 값 포함하여 적용하고 싶으면 dropna 인자값을 False로 입력)
- 넘파이 ndarray, 리스트, 딕셔너리와 DataFrame 간 상호 변환 => 교재 p.49 ~ p.53 참고
- [] : DataFrame의 칼럼 데이터 세트 생성과 수정
- drop() : DataFrame에서 데이터 삭제
  - axis 파라미터 값에 따라 특정 칼럼 또는 행 드롭
  - axis 0은 로우 방향 축, axis 1은 칼럼 방향 축
  - 기존 칼럼 값 가공해 새로운 칼럼 만들고 삭제하는 경우는 axis=1로 설정하고 드롭, 이상치 데이터 삭제하는 경우는 axis=0으로 설정하고 로우 레벨로 삭제
  - 여러 개의 칼럼 삭제하고 싶으면 리스트 형태로 삭제하고자 하는 칼럼명 입력해 labels 파라미터로 입력
  - 원본 DataFrame은 유지하고 드롭된 DataFrame을 새롭게 객체 변수로 받고 싶다면 inplace=False 로 설정
  - 원본 DataFrame에 드롭된 결과 적용할 경우에는 inplace=True 적용
- Index 객체
  - 판다스의 Index 객체는 RDBMS의 PK와 유사하게 DataFrame, Series의 레코드를 고유하게 식별하는 객체

- 한번 만들어진 DataFrame 및 Series의 Index 객체는 함부로 변경할 수 없음
- Series 객체는 Index 객체 포함하지만 Series 객체에 연산 함수 적용할 때 Index는 연산에서 제외되고 오직 식별용으로만 사용
- DataFrame 및 Series에 `reset_index()` 메서드 수행하면 새롭게 인덱스를 연속 숫자형으로 할당하며 기존 인덱스는 'index'라는 새로운 칼럼명으로 추가
- `reset_index()` : 인덱스가 연속된 int 숫자형 데이터 아닐 경우 연속 int 숫자형 데이터로 만들 때 주로 사용

- 데이터 셀렉션 및 필터링
  - 넘파이에서 `[]` 연산자는 행의 위치, 열의 위치, 슬라이싱 범위 등을 지정해 데이터를 가져올 수 있었지만 DataFrame 바로 뒤에 있는 `[]` 안에 들어갈 수 있는 것은 칼럼명 문자 또는 인덱스로 변환 가능한 표현식 (DataFrame 뒤에 있는 []는 칼럼만 지정할 수 있는 칼럼 지정 연산자)
  - 판다스의 인덱스 형태로 변환 가능한 표현식은 [] 내에 입력 가능
  - [] 내의 불린 인덱스 표현 가능
  - DataFrame 바로 뒤의 [] 연산자는 넘파이의 []나 Series의 []와 다름
  - DataFrame 바로 뒤의 [] 내 입력값은 칼럼명(또는 칼럼의 리스트)을 지정해 칼럼 지정 연산에 사용하거나 불린 인덱스 용도로만 사용해야 함
  - DataFrame[0:2]와 같은 슬라이싱 연산으로 데이터를 추출하는 방법은 사용하지 않는 게 좋음
  - 넘파이의 `[]` 연산자 = 판다스의 `iloc[]`, `loc[]` 연산자

- `iloc[]` : 위치(Location) 기반 인덱싱 방식으로 동작
  - 행과 열 위치를 0을 출발점으로 하는 세로축, 가로축 좌표 정숫값으로 지정
  - 행과 열의 좌표 위치에 해당하는 값으로 정숫값 또는 정수형의 슬라이싱, 팬시 리스트 값 입력해줘야 함
  - DataFrame의 인덱스 값이나 칼럼명 입력 시 오류 발생
  - 열 위치에 -1 입력하여 DataFrame의 가장 마지막 열 데이터 가져오는 데 자주 사용

	반환 값: 'Eunkyung'
data_df.iloc[2,1]	세번째 행의 두번째 열 위치에 있는 단일 값 반환 반환 값: '2015'
data_df.iloc[0:2, [0,1]]	0:2 슬라이싱 범위의 첫번째에서 두번째 행과 첫번째, 두번째 열에 해당하는 DataFrame 반환 반환 값: Name Year one Chulmin 2011 two Eunkyung 2016
data_df.iloc[0:2, 0:3]	0:2 슬라이싱 범위의 첫번째에서 두번째 행의 0:3 슬라이싱 범위의 첫번째부터 세번째 열 범위에 해당하는 DataFrame 반환 반환 값: Name Year Gender one Chulmin 2011 Male two Eunkyung 2016 Female
data_df.iloc[:]	전체 DataFrame 반환 Name Year Gender one Chulmin 2011 Male two Eunkyung 2016 Female three Jinwoong 2015 Male four Soobeom 2015 Male
data_df.iloc[:, :]	전체 DataFrame 반환 Name Year Gender one Chulmin 2011 Male two Eunkyung 2016 Female three Jinwoong 2015 Male four Soobeom 2015 Male

- `loc[]` : 명칭(Label) 기반 인덱싱 방식으로 동작
  - 데이터 프레임의 인덱스 값으로 행 위치를, 칼럼의 명칭으로 열 위치를 지정
  - 일반적으로 슬라이싱을 '시작값: 종료 값'과 같이 지정하면 시작 값 ~ 종료 값-1까지의 범위를 의미하지만, `loc[]` 에 슬라이싱 기호 적용하면 종료 값까지 포함하는 것 의미

'Name']	반환 값: 'Jinwoong'
data_df.loc['one':'two', ['Name', 'Year']]	인덱스 값 one부터 two까지 행의 Name과 Year 칼럼에 해당하는 DataFrame 반환 반환 값: Name Year one Chulmin 2011 two Eunkyung 2016
data_df.loc['one':'three', 'Name':'Gender']	인덱스 값 one부터 three까지 행의 Name부터 Gender 칼럼까지의 DataFrame 반환 반환 값: Name Year Gender one Chulmin 2011 Male two Eunkyung 2016 Female three Jinwoong 2015 Male
data_df.loc[:]	모든 데이터 값: Name Year Gender one Chulmin 2011 Male two Eunkyung 2016 Female three Jinwoong 2015 Male four Soobeom 2015 Male
data_df.loc[data_df.Year >= 2014]	iloc[ ]와 다르게 loc[ ]는 불린 인덱싱이 가능. Year 칼럼의 값이 2014 이상인 모든 데이터를 불린 인덱싱으로 추출 Name Year Gender two Eunkyung 2016 Female three Jinwoong 2015 Male four Soobeom 2015 Male

- 불린 인덱싱
  - iloc나 loc와 같이 명확히 인덱싱 지정하는 방식보다 불린 인덱싱에 의존해 데이터 가져오는 경우가 더 많음
  - 명칭이나 위치 지정 인덱싱에서 가져올 값은 주로 로직이나 조건에 의해 계산한 뒤에 행 위치, 열 위치 값으로 입력되는데, 그럴 필요 없이 처음부터 가져올 값을 조건으로 [] 내에 입력하면 자동으로 원하는 값을 필터링하기 때문
  - 불린 인덱싱은 `[]`, `loc[]` 에서 공통으로 지원! `iloc[]` 는 정수형 값 아닌 불린 값에 대해서는 지원하지 않으므로 불린 인덱싱 지원 X

- `sort_values()` : DataFrame과 Series의 정렬에 이용하는 메서드
  - RDBMS SQL의 order by 키워드와 유사
  - 주요 입력 파라미터는 by, ascending, inplace
  - by로 특정 칼럼 입력 시 해당 칼럼으로 정렬 수행
  - `ascending=True` 로 설정 시 오름차순으로 정렬, `ascending=False` 는 내림차순으로 정렬
  - `inplace=False` 로 설정 시 `sort_values()` 호출한 DataFrame은 그대로 유지하며

- aggregation 함수
  - `min()`, `max()`, `sum()`, `count()`
  - RDBMS SQL의 aggregation 함수 적용과 유사하지만 DataFrame의 경우 DataFrame에서 바로 aggregation 호출할 경우 모든 칼럼에 해당 aggregation 적용
  - 특정 칼럼에 aggregation 함수 적용하기 위해서는 DataFrame에 대상 칼럼들만 추출해 aggregation 적용

- `groupby()`
  - DataFrame에서 사용 시 입력 파라미터 by에 칼럼 입력하면 대상 칼럼으로 groupby 됨
  - DataFrame에 호출 시 DataFrameGroupBy라는 또 다른 형태의 DataFrame 반환
  - DateFrame에 호출해 반환된 결과에 aggregation 함수 호출 시 `groupby()` 대상 칼럼 제외한 모든 칼럼에 해당 aggregation 함수 적용
  - SQL의 경우 서로 다른 aggregation 함수를 적용할 경우에는 Select 절에 나열하기만 하면 되지만, DataFrame `groupby()` 의 경우 적용하려는 여러 개의 aggregation 함수명을 DataFrameGroupBy 객체의 `agg()` 내에 인자로 입력해서 사용
  - `groupby()` 를 이용해 API 기반으로 처리하다 보니 SQL의 group by보다 유연성이 떨어질 수밖에 없음

- 결손 데이터 처리
  - 결손 데이터: 칼럼에 값 없는, 즉 NULL인 경우 => 넘파이의 NaN으로 표시
  - `insa()` : NaN 여부 확인하는 API
  - `fillna()` : NaN 값을 다른 값으로 대체하는 API
  - `fillna()` 를 이용해 반환 값 다시 받거나 `inplace=True` 파라미터를 추가해야 실제 데이터 세트 값이 변경된다는 점 주의

- 판다스는 apply 함수에 lambda 식 결합해 DataFrame이나 Series의 레코드별로 데이터 가공하는 기능 제공

진규빈