【 CH 01. 파이썬 기반의 머신러닝과 생태계 이해 】

〔01. 머신러닝의 개념〕(1p.)

머신러닝: 애플리케이션[1]을 수정하지 않고도 데이터를 기반으로 패턴을 학습하고 결과를 예측하는 알고리즘 기법

(특징)

- 머신러닝은 급변하는 환경 속에서 기존의 소프트웨어 코드만으로는 해결하기 어려운 복잡한 문제를 효율적으로 처리할 수 있음. Ex) 금융 사기 거래 적발

- 머신러닝 알고리즘은 데이터 내의 패턴을 스스로 인지하고 데이터 및 수학적 기법을 기반으로 신뢰도 있는 예측 결과를 도출함. Ex) 스팸메일 분류

머신러닝의 분류

1. 지도학습: 분류, 회귀, 추천 시스템, 시각/음성 인지, 텍스트 분석, NLP(자연어처리)

2. 비지도학습: 클러스터링, 차원 축소, 강화학습

데이터 전쟁

머신러닝은 데이터에 매우 의존적이기 때문에, 최적의 머신러닝 알고리즘을 구축하는 것만이 아니라 좋은 품질의 데이터를 갖출 수 있는 능력도 중요함.

파이썬과 R 기반의 머신러닝 비교

파이썬과 R은 대표적인 머신러닝 프로그램 작성 언어.

- (R)

통계 전용 프로그램 언어 -> 통계 분석에 특화된 현업 사용자에 추천

- (파이썬)

---

[1] 애플리케이션: 운영 체제에서 실행되는 모든 소프트웨어. Ex) 워드프로세서, 웹브라우저, 컴파일러 등

다양한 영역에서 사용되는 개발 전문 프로그램 언어 -> 머신러닝 초보자, 개발자에게 추천

1. 쉽고 직관적 문법, 다양한 라이브러리 -> 높은 생산성

2. 유연한 프로그램 아키텍처 -> 객체[2]지향, 함수형 프로그래밍 모두 포괄

3. 인터프리터 언어[3]의 특성인 확장성, 유연성, 호환성 -> 서버, 네트워크, 시스템, IOT, 데스크톱 등 다양한 영역에서 사용

4. 머신러닝 애플리케이션과 결합한 다양한 애플리케이션 개발 가능

↘딥러닝 프레임워크가 파이썬을 중심으로 발전될 가능성이 큼

〔02. 파이썬 머신러닝 생태계를 구성하는 주요 패키지〕(6p.)

1. 머신러닝 패키지

- 사이킷런(Scikit-Learn): 데이터 마이닝[4]기반의 머신러닝으로 넘파이를 기반으로 작성됨.

2. 행렬/선형대수/통계 패키지

- 넘파이(NumPy): 행렬 기반의 데이터 처리에 특화돼 있어서 일반적 데이터 처리는 부족함

- 사이파이(SciPy): 자연학과 통계를 위한 다양한 패키지 소유

3. 데이터 핸들링

- 판다스: 대표적인 데이터 처리 패키지. 2차원 데이터 처리에 특화돼 있으며 넘파이보다 편리하게 데이터 처리를 할 수 있는 많은 기능 제공. 맷플롯립을 호출해 쉽게 시각화 기능을 지원 가능

4. 시각화

- 맷플롯립(Matplotlib): 대표적인 시각화 패키지. 그러나 너무 세분화된 API[5]로 익히기가 번거로우며 단순히 시각화를 위한 코드가 길어져 비효율적임.

---

[2] 프로그램에서 저장되는 모든 데이터는 객체이며, 각 객체는 신원(identity), 타입(클래스)과 값을 가짐.

[3] 인터프리터 언어: 소스코드를 한 줄 한 줄 읽어가며 명령을 바로 처리하는 프로그램(언어). 번역과 실행이 동시에 이루어짐. 반면, 컴파일 언어는 소스코드를 한꺼번에 다른 목적 코드로 번역한 후, 한 번에 실행함.

[4] 데이터 마이닝: 결과를 예측하기 위해 대량의 데이터 세트에서 이상점(anomalies)과 패턴 및 상관 관계를 찾아내는 프로세스

[5] **A**pplication **P**rogramming **I**nterface: 운영체제와 응용프로그램 사이의 통신에 사용되는 언어나 메시지 형식으로 중간 전달자 역할을 함.

- 시본(Seaborn): 맷플롯립의 문제점을 보완하기 위한 시각화 패키지. 더 함축적인 API, 다양한 유형의 그래프/차트 제공이 특징. 맷플롯립의 API를 이용해 함축적으로 만든 라이브러리이나 세밀한 부분의 제어는 여전히 맷플롯립의 API를 그대로 사용.

↘파이썬 기반의 머신러닝 개발을 위해서는 넘파이와 판다스에 대한 이해가 중요. 넘파이와 판다스는 사이킷런의 머신러닝 알고리즘에 사용되는 데이터 처리 대부분을 담당하기 때문.

〔03. 넘파이〕(13p.)

넘파이(**Num**ercial **Py**thon): C언어 기반의 전문 개발자가 만든 수치 계산에 있어 매우 빠른 연산을 수행하도록 만든 수치 계산 라이브러리

(특징)

- 루프를 사용하지 않고 대량 데이터의 배열 연산을 가능하게 하므로 배열 연산 속도가 빠름.

- 파이썬의 경우는 array 파트가 없고, list만 지원되기 때문에 배열 간의 연산 등은 넘파이를 주로 사용

- 넘파이는 여러 타입을 담을 수 있어서 상대적으로 느린 파이썬 리스트의 단점을 극복하기 위해 동일한 타입(Ex) 숫자-숫자, 문자-문자)에 대해서만 저장하여 속도를 빠르게 개선

넘파이 ndarray 개요

- 넘파이 기반 데이터 타입은 ndarray[6].

* np.array( ): 파이썬의 리스트와 같이 변환을 원하는 객체를 인자로 입력 받아서 array(배열)로 반환하는 함수

* ndarray.shape: ndarray(해당 배열)의 모양(차원과 크기)을 튜플 형태로 보여줌

* ndarray.ndim[7]: ndarray의 차원을 숫자 형태로 보여줌

Ex)

1. [1,2,3]인 array1의 shape = (3,) -> 1차원 array로 3개의 데이터를 가지고 있다는 의미

2. [[1,2,3],[2,3,4]]인 array2의 shape = (2,3) -> 2차원 array로, 2개의 로우와 3개의 칼럼으로 구성되

---

[6] ndarray: 다차원배열

[7] Ndim: 해당 배열의 차원

어 2*3=6개의 데이터를 가지고 있다는 의미

3. [[1,2,3]]인 array3의 shape = (1,3) -> 1개의 로우와 3개의 칼럼으로 구성된 2차원 데이터를 의미

ndarray의 데이터 타입

- ndarray 내의 데이터 값은 숫자, 문자열, 불 등이 모두 가능. 이때 ndarray 내의 데이터는 같은 타입과 형태여야 함. Ex) 한 개의 ndarray 객체에 int와 float이 같이 있을 수 없으며 [1 , 2, 3], [4, 5]도 불가.

* ndarray.dtype: ndarray 내의 데이터 타입 확인 or 리스트 -> 넘파이 과정에서 데이터 타입 변환

- 다른 데이터 유형이 섞여 있는 리스트를 ndarray로 변환하면 데이터 크기가 더 큰 데이터 타입으로 형 변환을 일괄 적용함. Ex) int -> 유니코드 문자열, int -> float

* ndarray.astype('타입'): ndarray 내의 데이터 타입 변경 <- 보통 대용량 데이터의 ndarray를 만들 때 메모리를 절약하기 위해 사용

ndarray를 편리하게 생성하기 – arrange, zeros, ones

* np.arrange(): array를 range()로 표현하는 것으로 0부터 함수 인자 값 -1까지의 값을 순차적으로 ndarray의 데이터 값으로 변환해 줌. 이때 default 함수 인자는 stop 값이지만, start 값도 부여할 수 있어 0이 아닌 다른 값부터 시작 가능

* zeros(): 함수 인자에 튜플 형태의 shape 값을 입력하면 모든 값을 0으로 채운 해당 shape를 가진 ndarray를 반환함.

* ones(): 0 대신 1

- 이때 함수 인자로 dtype을 정해주지 않으면 default는 float64 형

ndarray의 차원과 크기를 변경하는 reshape()

* ndarray.reshape(,): ndarray를 특정 차원 및 크기로 변환

- -1을 인자로 사용하면 원래 ndarray와 호환되는 새로운 shape로 변환해줌.

Ex) array1은 1차원 ndarray로 0~9까지 -> array1.reshape(-1, 5): array1과 호환될 수 있는 2차원 ndarray로 변환하되, 고정된 5개의 칼럼(열)에 맞는 로우(행)를 생성. 즉, 10개의 1차원 데이터와 호환될 수 있는 고정된 5개 칼럼에 맞는 로우 개수는 2이므로 2 x 5의 2차원 ndarray로 변환

- -1인자는 reshape(-1, 1)와 같은 형태로 자주 이용됨. Reshape(-1, 1)은 원본 ndarray가 어떤 형태

라도 2차원이고, 여러 개의 로우를 가지되 반드시 1개의 칼럼을 가진 ndarray로 변환됨.

* ndarray.tolist(): 리스트 자료형으로 변환

넘파이의 ndarray의 데이터 세트 선택하기 – 인덱싱(Indexing)

1. 단일 값 추출: 원하는 위치의 인덱스 값 지정 -> 해당 위치의 데이터가 반환됨

- axis 0: 로우 방향 축 / axis 1: 칼럼 방향 축 / axis 2: 높이 방향 축

- 넘파이에서는 로우와 칼럼 대신 축 사용 Ex) [row=0, col=1] -> [axis 0=0, axis 1=1]

- 축 기반의 연산에서 axis가 생략되면 axis 0을 의미

2. 슬라이싱(Slicing): 연속된 인덱스 상의 ndarray를 추출

- 단일 데이터 값 추출을 제외하고 슬라이싱, 팬시 인덱싱, 불린 인덱싱으로 추출된 데이터 세트는 모두 ndarray 타입

3. 팬시 인덱싱(Fancy Indexing): 일정한 인덱싱 집합을 리스트 또는 ndarray 형태로 지정해 해당 위치에 있는 데이터의 ndarray를 반환

4. 불린 인덱싱(Boolean Indexing): 특정 조건 해당 여부인 True/False 값 인덱싱 집합을 기반으로 True에 해당하는 인덱스 위치에 있는 데이터의 ndarray를 반환

* ndarray[조건]: 조건 필터링과 검색이 동시에 가능. for loop보다 간편

불린 인덱싱 동작 단계

1. array1d >5와 같이 ndarray의 필터링 조건을 [] 안에 기재

2. False 값은 무시하고 True 값에 해당하는 인덱스값만 저장(주의: True 값 자체인 1을 저장하는 것이 아니라 True 값을 가진 인덱스를 저장함)

3. 저장된 인덱스 데이터 세트로 ndarray 조회

행렬의 정렬 – sort()와 argsort()

<행렬 정렬>

- 행렬 정렬에는 np.sort()와 같이 넘파이에서 sort()를 호출하는 방식과 ndarray.sort()와 같이 행렬 자체에서 sort()를 호출하는 방식이 있음.

- np.sort()의 경우 원 행렬은 그대로 유지한 채 원 행렬의 정렬된 행렬을 반환하며, ndarray.sort()는 원 행렬 자체를 정렬한 형태로 변환하며 반환 값은 None임.

<정렬된 행렬의 인덱스 반환하기>

원본 행렬이 정렬되었을 때 기존 원본 행렬의 원소에 대한 인덱스를 필요로 할 때 np.argsort()를 이용함. np.argsort()는 정렬 행렬의 우너본 행렬 인덱스를 ndarray 형으로 반환함.

선형대수 연산 – 행렬 내적과 전치 행렬 구하기

행렬 내적(행렬 곱): 행렬 내적은 행렬 곱이며, 두 행렬 A와 B의 내적은 왼쪽 행렬의 로우(행)와 오른쪽 행렬의 칼럼(열)의 원소들을 순차적으로 곱한 뒤 그 결과를 모두 더한 값으로 np.dot()을 이용해 계산 가능.

전치 행렬: 원 행렬에서 행과 열 위치를 교환한 원소로 구성한 행렬. 넘파이의 transpose()를 이용해 구할 수 있음

〔04. 데이터 핸들링 - 판다스〕(39p.)

판다스

- 행과 열로 이뤄진 2차원 데이터를 효율적으로 가공/처리할 수 있는 다양한 기능 제공.

- 넘파이 기반으로 작성되었지만, 저수준 API가 대부분인 넘파이와 달리, 훨씬 유연하고 편리한 데이터 핸들링이 가능함.

- 파이썬의 리스트, 컬렉션, 넘파이 등의 내부 데이터뿐만 아니라 CSV 등의 파일을 쉽게 DataFrame으로 변경해 데이터의 가공/분석을 편리하게 함.

* DataFrame: 판다스의 핵심 객체로 여러 개의 행과 열로 이뤄진 2차원 데이터를 담는 데이터 구조체. 여러 개의 Series로 이뤄짐.

* Series: DataFrame과 달리 칼럼이 하나뿐인 데이터 구조체.

* Index: RDBMS의 PK처럼 개별 데이터를 고유하게 식별하는 Key 값. Series와 DataFrame 모두 Index를 Key 값으로 가지고 있음.

판다스 시작 – 파일을 DataFrame으로 로딩, 기본 API

판다스는 다양한 포맷으로 된 파일을 DataFrame으로 로딩할 수 있는 편리한 API를 제공. 대표적

으로 read_csv(), read_table(), read_fwf()가 있음.

- read csv(): CSV(칼럼을 ','로 구분한 파일 포맷) 파일 포맷 변환을 위한 API.

CSV뿐만 아니라 어떤 필드 구분 문자 기반의 파일 포맷이라도 read_csv()의 인자인 sep에 해당 구분 문자 입력을 통해 DataFrame으로 변환 가능. Ex) 탭 구분 -> read_csv('파일명', sep = '₩t')

이때 read_csv에서 sep 인자를 생략하면 자동으로 콤마로 할당. Ex) (sep = ',')

- read_table(): 필드 구분 문자(Delimeter)가 탭('₩t'). read csv()와 기능상 큰 차이 없음

- read_fwf(): 필드 구분 문자가 콤마(',')

- read_fwf(): Fixed Width, 즉 고정 길이 기반의 칼럼 포맷을 DataFrame으로 로딩하기 위한 API.

- read_csv(filepath_or_buffer, sep = ', ',...) 함수에서 가장 중요한 인자는 filepath로 로드하려는 데이터 파일의 경로를 포함한 파일명을 입력하면 됨.