01 머신러닝 개념

- 정의
  - 데이터를 기반으로 통계적인 신뢰도를 강화하고 예측 오류를 최소화하기 위한 다양한 수학적 기법을 적용해 데이터 내의 패턴을 스스로 인지(학습)하고 신뢰도 있는 예측 결과를 도출
- 분류
  - 지도학습**(Supervised Learning)**
    - 분류**(Classification)**
    - 회귀**(Regression)**
    - 추천 시스템
    - 시각/음성 감지/ 인지
    - 텍스트 분석, NLP
  - 비지도학습**(Un-supervised Learning)**
    - 클러스터링
    - 차원 축소
    - 강화학습
  - 강화학습**(Reinforcement Learning)**
- 데이터의 중요성
  - 어떤 품질의 데이터로 만든 머신러닝 모델인지에 따라 머신러닝의 수행 결과 및 성능이 좌우됨
  - 데이터를 이해하고 효율적으로 가공, 처리, 추출해 최적의 데이터를 기반으로 알고리즘을 구동할 수 있도록 준비하는 능력의 중요성 커짐

02 파이썬 머신러닝 생태계를 구성하는 주요 패키지

- 머신러닝 패키지
  - 사이킷런**(Scikit-Learn)**
    - 데이터 마이닝 기반의 머신러닝에서 독보적 위치 차지
  - 텐서플로**(TensorFlow)**, 케라스**(Keras)** 등 전문 딥러닝 라이브러리
    - 영상, 음성, 언어 등의 비정형 데이터 분야
- 행렬, 선형대수, 통계 패키지
  - 넘파이**(NumPy)**
    - 행렬과 선형대수를 다루는 패키지, 행렬 기반의 데이터 처리에 특화(= 일반적인 데이터 처리에 부족)
  - 사이파이**(SciPy)** - 자연과학과 통계를 위한 다양한 패키지
- 데이터 핸들링
  - 판다스**(Pandas)**
    - 데이터 처리 패키지, 2차원 데이터 처리에 특화, 맷플롯립(Matplotlib)을 호출해 시각화 기능 지원 가능
- 시각화
  - 맷플롯립**(Matplotlib)**
    - 파이썬의 대표적인 시각화 라이브러리
    - 세분화된 API로 익히기 어렵

      ** API(Application Programming Interface) : 컴퓨터와 컴퓨터 프로그램(소프트웨어) 사이의 연결
  - 시본**(Seaborn)**
    - 맷플롯립을 보완하기 위한 시각화 패키지
    - 판다스와 더 쉬운 연동
- 아이파이썬**(IPython, Interactive Python)**
  - 아이파이썬(IPython)
    - 대화형 파이썬 툴
    - 전체 프로그램에서 특정 코드 영역별로 개별 수행 지원하므로 영역별로 코드 이해가 명확하게 설명 가능

      ex. 주피터 노트북(Jupyter notebook)

03 넘파이(Numerical Python, NumPy)

- 넘파이 특징
  - 빠른 배열 연산 속도
  - C/C++과 같은 저수준 언어 기반의 API 제공
  - 다양한 데이터 핸들링 가능
- **ndarray**
  - 넘파이의 기반 데이터 타입
  - ndarray를 통해 넘파이에서 다차원 배열을 쉽게 생성하고 다양한 연산 수행 가능
- **np.array(ndarray로 변환을 원하는 객체)**
  - 파이썬의 리스트와 같은 다양한 인자 입력 받음

    → ndarray로 변환

    ** 리스트의 경우 [ ] : 1차원, [[ ]] : 2차원

ex) np.array([[1,2,3]])

- **array명.shape**
  - ndarray의 행과 열의 수를 튜플 형태로 가짐
  - ndarray의 차원 알 수 있음
- **array명.ndim**
  - array의 차원을 알려줌
- ndarray의 데이터 타입
  - ndarray 내의 데이터 값은 숫자, 문자열, 불 등 모두 가능
  - ndarray 내의 데이터 타입은 같은 데이터 타입만 가능
  - 만약 다른 데이터 유형이 섞여 있는 리스트를 ndarray로 변경한다면 데이터 크기가 더 큰 데이터 타입으로 형 변환을 일괄 적용
- **array명.astype(원하는 타입을 문자열로 지정)**
  - ndarray 내 데이터값의 타입 변경
  - 메모리를 더 절약해야 할 때 보통 이용

  ex) array1.astype('float64')
- **np.arange(stop 값)**
  - 파이썬 표준 함수인 range( )와 유사 기능
  - array를 range로 표현
  - 0부터 함수 인자값-1까지의 값을 순차적으로 ndarray의 데이터 값으로 변환
  - range와 유사하게 start 값도 부여해 0이 아닌 다른 값부터 시작한 연속 값 부여 가능

  ex) np.arange(10)
- **np.zeros(튜플 형태의 shape 값)**
  - 모든 값을 0으로 채운 해당 shape을 가진 ndarray 반환

  ex) np.zeros((3,2))
- **np.ones(튜플 형태의 shape 값)**
  - 모든 값을 1로 채운 해당 shape을 가진 ndarray 반환
  - 함수 인자로 dtype을 정해주지 않으면 default 값으로 float64형의 데이터로 ndarray 채움

  ex) np.ones((2,3),dtype = 'int32')
- **array명.reshape(행, 열)**
  - ndarray를 특정 차원 및 크기로 변환
  - 지정된 사이즈로 변경이 불가능하면 오류 발생

  ex) (10,)인 데이터를 (4,3) shape 형태로 변경 불가
  - 인자로 -1을 사용하면 원래 ndarray와 호환되는 새로운 shape로 변환

  ** 물론 -1을 사용하더라도 호환할 수 없는 형태는 변환 불가

  ex) array1.reshape(2,5)

  ex) array1.reshape(2,2,2)

- **array명.reshape(-1, 1)**
  - 원본 ndarray가 어떤 형태라도 2차원이고 여러 행을 가지되 반드시 1개의 열을 가진 ndarray로 변환
- 인덱싱**(Indexing)**
  - 특정한 데이터만 추출
    - 원하는 위치의 인덱스 값을 저장하면 해당 위치의 데이터가 반환됨
    - axis 0 : 행 방향의 축
    - axis 1 : 열 방향의 축

  ex) array1[0], array2[1,0]
- 슬라이싱**(Slicing)**
  - 연속된 인덱스상의 ndarray 추출하는 방식
  - ' 시작 : 끝 ' → 시작 인덱스부터 끝-1 위치에 있는 데이터의 ndarray를 반환

  **시작/끝 인덱스 생략 가능

  **n차원 ndarray에서 뒤에 오는 인덱스 없애면 n-1차원 ndarray 반환(n>1)

  ex) array1[0:5], array1[:],
  array2[0:2,1:3], array2[0]
- 팬시 인덱싱**(Fancy Indexing)**
  - 일정한 인덱싱 집합을 리스트 또는 ndarray 형태로 지정해 해당 위치에 있는 데이터의 ndarray를 반환

  ex) array2[[0,1],2]]

  → (0,2), (1,2)로 적용됨
- 불린 인덱싱**(Boolean Indexing)**
  - 특정 조건에 해당하는지 여부인 True/False 값 인덱싱 집합을 기반으로 True에 해당하는 인덱스 위치에 있는 데이터를 ndarray로 반환

  ex) array1[array1 > 5]
- **np.sort(정렬하고 싶은 행렬)**

- 넘파이에서 sort( )를 호출하는 방식
- 원 행렬은 그대로 유지한 채 원 행렬의 정렬된 행렬을 반환
  **내림차순으로 정렬하기 위해서는 [::-1]을 적용
  **행렬이 2차원 이상일 경우 axis 축 값 설정을 통해 행/열 방향으로 정렬 수행 가능
  ex) np.sort(array1),
  np.sort(array1)[::-1],
  np.sort(array2, axis = 0)

- array명.sort( )
  - 행렬 자체에서 sort( )를 호출하는 방식
  - 원 행렬 자체를 정렬한 형태로 변환, 변환 값은 None
    ex) array1.sort( )
- np.argsort(정렬하고 싶은 행렬)
  - 원본 행렬이 정렬되었을 때 기존 원본 행렬의 원소에 대한 인덱스를 ndarray형으로 반환
  - 내림차순으로 정렬 시에 원본 행렬의 인덱스를 구할 경우 np.argsort( )[::-1] 적용
  - 넘파이의 ndarray는 판다스 DataFrame 열과 같은 메타 데이터를 가질 수 없음
    → 실제 값과 그 값이 뜻하는 메타 데이터를 별도의 ndarray로 각각 가져야 함
    ex) np.argsort(array1),
    np.argsort(array1)[::-1],
    시험 성적순으로 학생 이름을 출력하고자 한다면
    score_index = np.argsort(score_array)
    name_array[scroe_index]로 출력 가능
- np.dot(행렬1, 행렬2)
  - 행렬 내적(행렬 곱) 구하기
    ex) np.dot(A, B)
- np.transpose(행렬1)
  - 전치 행렬 구하기
    ** 전치 행렬 : 원 행렬에서 행과 열의 위치를 교환한 원소로 구성한 행렬
    ex) np.transpose(A)

- 판다스
  - 파이썬에서 데이터 처리를 위해 존재하는 가장 인기 있는 라이브러리
- DataFrame
  - 판다스의 핵심 개체
  - 여러 개의 행과 열로 이뤄진 2차원 데이터를 담은 구조체
  - 여러개의 Series로 이뤄졌다고 할 수 있음
    ** Series : 열이 하나뿐인 구조체
- Index
  - RDBMS의 PK처럼 개별 데이터를 고유하게 식별하는 Key값
  - Series와 DataFrame은 모두 Index를 Key값으로 가짐
- pd.read_csv('파일 경로')
  - 호출 시 파일명 인자로 들어온 파일을 로딩해 DataFrame 객체로 반환
- 데이터프레임명.head(N)
  - 맨 앞에 있는 N개의 행을 반환
    ** default는 5개
- 데이터프레임명.shape
  - 데이터프레임의 행과 열을 튜플 형태로 반환
- 데이터프레임명.info( )
  - 총 데이터 건수, 데이터 타입, Null 건수 알 수 있음
- 데이터프레임명.describe( )
  - 열별 숫자형 데이터값의 n-percentile 분포도, 평균값, 최대값, 최솟값 나타냄
- 데이터프레임명['열의 이름'].value_counts( )
  - 해당 열값의 유형과 건수 확인 가능
  - 많은 건수 순서로 정렬되어 값 반환
    ** value_counts(dropna = False) : Null 값을 포함하여 건수 계산
    ex) value_counts = titanic_df['Pclass'].value_counts( )
- pd.DataFrame(변환할 인자)
  - 리스트, ndarray, 딕셔너리 등 데이터프레임으로 변환 가능
  - 열이름을 columns = 지정할 열이름을 통해 지정할 수 있음
    ** 딕셔너리의 경우 Key 값은 열이름, 각 Value 값은 열 데이터로 매핑됨

ex) pd.DataFrame(list1, columns = col_name1)

- 데이터프레임명**.values**
  - 데이터프레임을 ndarray로 변환
- 데이터프레임명**.tolist( )**
  - 데이터프레임을 리스트로 변환
- 데이터프레임명**.to_dict( )**
  - 데이터프레임을 딕셔너리로 변환
  - 인지로 **'list'**를 입력하면 딕셔너리의 값이 리스트형으로 반환됨

    ex) df_1.to_dict('list')

- 데이터프레임명**['새로운 열 이름'] =** 할당할 값
  - 새로운 열과 그에 해당하는 일괄적인 값을 할당할 수 있음

    ex) titanic_df['Age_0'] = 0

    ** Series에 상수값을 할당하면 Series의 모든 데이터 세트에 일괄적으로 적용됨

    ex) titanic_df['Age_0']*10
- 데이터프레임명**.drop( )**
  - 데이터프레임에서의 데이터 삭제
- **drop( )** 메서드의 원형
  - 데이터프레임명.drop(**labels = None, axis = 0**, index = None, columns = None, level = None, **inplace = False**, errors = 'raise')
  - **labels**에 원하는 열 이름을 입력, axis = 1하면 지정된 열 드롭
  - **axis** 값에 따라 특정 열 또는 행 드롭
  - **inplace = False** 자기 자신의 데이터프레임의 데이터는 삭제하지 않은채 삭제된 결과 데이터프레임 반환
  - **inplace = True** 자신의 데이터프레임의 데이터 삭제

    ex) drop_result = titanic_df.drop['Age_0', axis = 1, inplace = True)

    ** inplace = True로 설정한 채 반환값을 다시 자신의 데이터프레임 객체로 할당하면 자신의 데이터프레임 변수를 None으로 만들어버림

ex) titanic_df = titanic_df.drop[..., inplace = True (X)

- 데이터프레임명**.index**
  - index 객체 추출
  - index 객체는 식별성 데이터를 1차원 array로 가짐

- **Index**
  - ndarray와 유사하게 단일 값 반환 및 슬라이싱 가능

    ex) index 객체 변수명.values.shape, index 객체 변수명[:5].values, index 객체 변수명.values[:5], index 객체 변수명[6]

    ** index 객체의 값 변경 불가능

    ex) index 객체 변수명[0] = 5 (X)
  - Series 객체에 연산 함수를 적용할 때 Index는 연산에서 제외됨

    ex) Series 객체 변수명.max( )/sum( ), sum(Series 객체 변수명)
- 데이터프레임명**.reset_index(inplace = False)**
  - 새롭게 인덱스를 연속 숫자형으로 할당하며 기존 인덱스는 **'index'**라는 새로운 열이름으로 추가됨
- 데이터프레임명**[ ]**
  - 열만 지정할 수 있는 **'열 지정 연산자'**로 이해
- 데이터프레임명**.iloc[ ]**
  - 행과 열의 좌표 위치에 해당하는 값을 입력
  - **0**부터 시작, **-1**까지

    ex) titanic_df.iloc[0,2]

    titanic_df.iloc[:] - 전체 데이터프레임 반환

    titanic_df.iloc[:,:] - 전체 데이터프레임 반환

- 데이터프레임명**.loc[ ]**
  - 행의 인덱스, 열의 이름을 입력
    ex) data_df.loc[‘one’, ‘Name’]
  - 불린 인덱싱 가능
    ex) data_df.loc[data_df. Year >=2015]
- 불린 인덱싱

ex) titanic_df[titanic_df[‘Age’] > 60]
  - 60세 이상인 승객의 나이와 이름만 추출
    ex) titanic_df[titanic_df[‘Age’] > 60][[‘Name’, ‘Age’]],
    titanic_df.loc[titanic_df[‘Age’] > 60, [‘Name’, ‘Age’]]
  - 60세 이상, 선실 등급 1등급, 성별 여성 추출
    titanic_df[(titanic_df[‘Age’] > 60) & (titanic_df[‘Pclass’] == 1) & (titanic_df[‘Sex’] == ‘female’)],
    titanic_df[cond1 & cond2 & cond3]
- **sort_values( )**
  - by = [‘특정 열이름’] : 해당 열로 정렬 수행
    ex) titanic_df.sort_values(by = [‘Name’])
  - ascending = True : 오름차순으로 정렬(기본)
  - ascending = False : 내림차순으로 정렬
    ex) titanic_df.sort_values(by = [‘Pclass’,‘Name’], ascending = False)
  - inplace = False : 데이터프레임 유지 & 정렬된 결과를 반환(기본)
  - inplace = True : 호출한 데이터프레임의 정렬 결과 그대로 적용
- **Aggregation** 함수 적용
  - 모든 열에 해당 aggregation을 적용
  - 데이터프레임명.min( )/max( )/sum( )/ count( )
    ex) titanic_df.count( )
    titanic_df[[‘Age’, ‘Fare’]].mean( )
- 데이터프레임명**.groupby(‘**열 이름**’)**
  - 입력된 열 기준으로 groupby 됨
  - 집단, 그룹별로 데이터를 집계, 요약
  - groupby( )를 호출해 반환된 결과에 aggregation 함수를 호출하면 groupby( ) 대상 열을 제외한 모든 열에 해당 aggregation() 함수 적용
    ex) titanic_df.groupby(‘Pclass’)[[‘PassengerId’, ‘Survived’]].count( )
- 데이터프레임명**.isna( )**
  - 데이터가 NaN인지 아닌지 판별
  - 결손 데이터의 개수 구할 때 사용
    ex) titanic.isna( ).sum( )
- 데이터프레임명**.fillna( )**
  - 결손 데이터 대체하기
    ex) titanic_df[‘Age’].fillna(titanic_df[‘Age’].mean( ))

- **apply lambda**
  - lambda : 함수의 선언과 함수 내의 처리를 한 줄의 식으로 쉽게 변환
  - lambda 입력인자 : 입력인자를 기반으로 한 계산식으로 호출 시 계산 결과 반환됨
    ex) lambda x : x**2

01 사이킷런 소개와 특징

- 사이킷런**(scikit-learn)**
  - 쉽고 가장 파이썬스러운 API 제공
  - 머신러닝을 위한 다양한 알고리즘, 개발을 위한 편리한 프레임워크, API 제공
  - 매우 많은 환경에서 사용됨

02 붓꽃 품종 예측하기

- 지도학습**(Supervised Learning)**
  - 학습을 위한 다양한 피처와 분류 결정값인 레이블(Label) 데이터로 모델을 학습한 뒤, 별도의 테스트 데이터 세트에서 미지의 레이블 예측
  - 학습 데이터 세트 : 학습을 위해 주어진 데이터 세트
  - 테스트 데이터 세트 : 머신러닝 모델의 예측 성능을 평가하기 위해 별도로 주어진 데이터 세트
  - 대표적인 예 : 분류, 회귀
- **sklearn**
  - 사이킷런 패키지 내의 모듈명
- **sklearn.tree**
  - sklearn.tree 내의 모듈은 트리 기반 ML 알고리즘을 구현한 클래스의 모임
- **sklearn.model_selection**
  - 학습 데이터와 검증 데이터, 예측 데이터로 데이터를 분리하거나 최적의 하이퍼 파라미터로 평가하기 위한 다양한 모듈의 모임
- 하이퍼 파라미터
  - 머신러닝 알고리즘별로 최적의 학습을 위해 직접 입력하는 파라미터
  - 이를 통해 머신러닝 알고리즘의 성능 튜닝 가능
- **DecisionTreeClassifier**
  - 의사 결정 트리(Decision Tree) 알고리즘을 구현한 것
- **train_test_split( )**
  - 데이터 세트를 학습 데이터와 테스트 데이터로 분리

```
X_train, X_test, y_train, y_test =
train_test_split(iris_data, iris_label,
test_size = 0.2, random_state = 11)
```

  - X_train : 학습용 피처 데이터 세트
  - X_test : 테스트용 피처 데이터 세트
  - y_train : 학습용 레이블 데이터 세트
  - y_test : 테스트용 레이블 데이터 세트
  - iris_data : 피처 데이터 세트
  - iris_label : 레이블 데이터 세트
  - test_size : 전체 데이터 세트 중 테스트 데이터 세트의 비율
  - random_state : 호출할 때마다 같은 학습/테스트용 데이터 세트를 생성하기 위해 주어지는 난수 발생 값
- 데이터프레임명**.fit()**
  - 학습 수행
  - fit(학습용 피처 데이터 속성, 결정값 데이터 세트)

    ex) df_clf.fit(X_train, y_train)
- 데이터프레임명**.predict( )**
  - 학습한 모델 기반에서 테스트 데이터 세트에 대한 예측값 반환
  - predict(테스트용 피처 데이터 세트)

    ex) df_clf.predict(X_test)
- 정확도
  - 예측 결과가 실제 레이블 값과 얼마나 정확하게 맞는지를 평가하는 지표
- **accuracy_score( )**
  - 정화도 측정을 위한 사이킷런에서 제공하는 함수
  - accuracy_score(실제 레이블 데이터 세트, 예측 레이블 데이터 세트)

    ex) accuracy_score(y_test, pred

03 사이킷런의 기반 프레임워크 익히기

- **Estimator** 클래스
  - 지도학습의 모든 알고리즘을 구현한 클래스

    ex) Classifier, Regressor 클래스
- 사이킷런의 주요 모듈

- 예제 데이터
  - sklearn.datasets
    : 사이킷런에 내장되어 예제로 제공하는 데이터 세트
- 피처 처리
  - sklearn.preprocessing
    : 데이터 전처리에 필요한 다양한 가공 기능 제공(문자열을 숫자형 코드 값으로 인코딩, 정규화, 스케일링 등)
  - sklearn.feature_selection
    - 알고리즘에 큰 영향을 미치는 피처를 우선순위대로 셀렉션 작업을 수행하는 다양한 기능 제공
  - sklearn.feature_extraction
    - 텍스트 데이터나 이미지 데이터의 벡터화된 피처를 추출하는 데 사용됨
- 피처 처리 & 차원 축소
  - sklearn.decomposition
    - 차원 축소와 관련된 알고리즘을 지원하는 모듈
    - PCA, NMF, Truncated SVD 등을 통해 차원 축소 기능 수행 가능
- 데이터 분리, 검증 & 파라미터 튜닝
  - sklearn.model_selection
    - 교차 검증을 위한 학습용/테스트용 분리
    - 그리드 서치(Grid Search)로 최적 파라미터 추출 등의 API 제공
- 평가
  - sklearn.metrics
    - 분류, 회귀, 클러스터링, 페어와이즈(Pairwise)에 대한 다양한 성능 측정 방법 제공
    - Accuracy, Precision, Recall, ROC-AUC, RMSE 등 제공
- ML 알고리즘
  - sklearn.ensemble
    - 앙상블 알고리즘 제공
    - 랜덤 포레스트, 에이다부스트, 그래디언트 부스팅 등 제공
  - sklearn.linear_model
    - 주로 선형 회귀, 릿지(Ridge), 라쏘(Lasso) 및 로지스틱 회귀 등 회귀 관련 알고리즘을 지원
    - SGD(Stochastic Gradient Descent) 관련 알고리즘도 제공
  - sklearn.naive_bayes
    - 나이브 베이즈 알고리즘 제공
    - 가우시안 NB, 다항분포 NB emd
  - sklearn.neighbors
    - 최근접 이웃 알고리즘 제공
    - K-NN 등
  - sklearn.svm
    - 서포트 벡터 머신 알고리즘 제공
  - sklearn.tree
    - 의사 결정 트리 알고리즘 제공
  - sklearn.cluster
    - 비지도 클러스터링 알고리즘 제공
    - K-평균, 계층형, DBSCAN
- 유틸리티
  - sklearn.pipeline
    - 피처 처리 등의 변환과 ML 알고리즘 학습, 예측 등을 함께 묶어서 실행할 수 있는 유틸리티 제공
- 내장된 예제 데이터 세트(p.94)

04 Model Selection 모듈 소개

- **train_test_split( )**
  - 학습/테스트 데이터 분리
  - 선택적으로 입력받는 파라미터
    - test_size
      : 전체 데이터에서 데이터 세트 크기를 얼마로 샘플링할 것인지
      : 디폴트값 = 0.25
    - train_size
      : 전체 데이터에서 학습용 데이터 세트 크기를 얼마로 샘플링할 것인지
      : 주로 사용되지 않음
    - shuffle
      : 데이터를 분리하기 전 데이터를 미리 섞을지 결정
      : 디폴트값 = True

    - random_state
      : 호출할 때마다 동일한 학습/테스트용 데이터 세트를 생성하기 위해

- 과적합
  - 모델이 학습 데이터에만 과도하게 최적화되어, 실제 예측을 다른 데이터로 수행할 경우 예측 성능이 과도하게 떨어지는 경우
- 교차 검증
  - 과적합과 같은 문제점을 개선하기 위해
  - 데이터의 편중을 막기 위해 별도의 여러 세트로 구성된 학습 데이터 세트와 검증 데이터 세트에서 학습과 평가를 수행
  - 교차검증을 기반으로 1차 평가를 한 뒤 최종적으로 테스트 데이터 세트에 적용해 ML 모델 성능 평가
- 데이터 세트의 구분
  - 학습 데이터 세트
    - 학습 데이터 세트
    - 검증 데이터 세트
      : 최종 평가 이전에 학습된 모델을 다양하게 평가
  - 테스트 데이터 세트
    - 모든 학습/검증 과정이 완료된 후 최종적으로 성능을 평가하기 위한 데이터 세트
- **K** 폴드 교차 검증
  - K개의 데이터 폴드 세트를 만들어서 K번만큼 각 폴드 세트에 학습과 검증 평가를 반복적으로 수행하는 방법
    ex) KFold(n_split = 5)
  - KFold 객체의 split( )를 호출하면 폴드별 학습용, 검증용 테스트의 행 인덱스를 array로 반환
    ex) kfold.split(features)
  - KFold로 데이터를 학습하고 예측하는 과정
    - 폴드 세트를 설정
    - for 루프에서 반복적으로 학습 및 테스트 데이터의 인덱스 추출
    - 반복적으로 학습과 예측 수행, 예측 성능 반환

- **Stratified K** 폴드
  - 불균형한 분포도를 가진 레이블(결정 클래스) 데이터 집합을 위한 K 폴드 방식
    ** 불균형한 분포도를 가진 레이블 데이터 집합
    : 특정 레이블 값이 특이하게 많거나 매우 적어서 값의 분포가 한쪽으로 치우치는 것
  - K 폴드가 레이블 데이터 집합이 원본 데이터 집합의 레이블 분포를 학습 및 테스트 세트에 제대로 분배하지 못하는 경우의 문제 해결해줌
    ex) StratifiedKFold(n_splits = 3)
  - StratifiedKFold의 split( ) 호출 시 반드시 레이블 데이터 세트의 추가 입력 필요
    ex) skfold.split(features, label)
  - 회귀에서 Stratified K 폴드 지원 X
- **cross_val_score( )**
  - KFold로 데이터를 학습하고 예측하는 과정을 한꺼번에 수행해주는 API(교차검증 보다 편하게)
  - cross_val_score(estimator, X, y = None, scoring = None, cv = None, n_jobs = 1, verbose = 0, fit_params = None, pre_dispatch = '2*n_jobs')
  - 주요 파라미터
    - estimator
      : 사이킷런의 분류 알고리즘 클래스인 Classifier 또는 회귀 알고리즘 클래스인 Regressor 의미
    - X
      : 피처 데이터 세트
    - y
      : 레이블 데이터 세트
    - scoring

: 예측 성능 평가 지표

- cv

: 교차 검증 폴드 수

ex) cross_val_score(df_clf, data, label, scoring = 'accuracy', cv = 3)

- 수행 후 scoring 파라미터로 지정된 성능지표 측정값을 배열 형태로 반환
- cross_validate( )
  - cross_val_score( )과 비슷
  - 여러 개의 평가 지표 반환 가능
  - 학습 데이터에 대한 성능 평가 지표와 수행 시간 같이 제공

- **GridSearchCV**
  - 교차 검증과 최적 하이퍼 파라미터 튜닝을 한번에
  - 교차 검증을 기반으로 하이퍼 파라미터의 최적 값을 찾게 해줌
  - 수행시간이 상대적으로 오래 걸림
  - 주요 파라미터
    - estimator

      : classifier, regressor, pipeline이 사용될 수 있음
    - param_grid

      : key + 리스트 값을 가지는 딕셔너리 주어짐

      : estimator의 튜닝을 위해 파라미터명과 사용될 여러 파라미터 값 지정
    - scoring

      : 예측 성능을 측정할 평가 방법 지정
    - cv

      : 교차 검증을 위해 분할되는 학습/테스트 세트의 개수 지정
    - refit

      : 디폴트 = True

      : True로 생성 시 가장 최적의 하이퍼 파라미터를 찾은 뒤 입력된 estimator 객체를 해당 하이퍼 파라미터로 재학습

ex) GridSearchCV(dtree, param_grid = parameters, cv = 3, refit = True)

05 데이터 전처리

- 데이터 인코딩
  - 모든 문자열 값을 인코딩하여 숫자형으로 변환
  - 문자열 피처
    - 카테고리형 피처
    - 텍스트형 피처

- 레이블 인코딩(**Label Encoding**)
  - 카테고리 피처를 코드형 숫자 값으로 변환

    ex) TV : 1, 냉장고 : 2, 전자레인지 : 3, 컴퓨터 : 4, 선풍기 : 5, 믹서 : 6 과 같은 숫자형 값으로 변환
  - 레이블 인코딩이 일괄적인 숫자 값으로 변환이 되면서 몇몇 ML 알고리즘에는 이를 적용할 경우 예측 성능이 떨어지는 경우 발생 가능

    → 선형회귀과 같은 알고리즘에는 적용 X
  - 숫자 값의 경우 크고 작음에 대한 특성이 작동하기 때문

    ex) 2인 냉장고가 1인 TV보다 큰 값이므로 특정 ML 알고리즘에서는 가중치가 더 부여되거나 더 중요하게 인식할 가능성 발생
  - 코드

    ex)

    encoder = LabelEncoder( )

    encoder.fit(items)

    labels = encoder.transform(items)
- 원-핫 인코딩(**One-Hot Encoding**)
  - 피처 값의 유형에 따라 새로운 피처를 추가해 고유 값에 해당하는 칼럼에만 1을 표시하고 나머지 칼럼에는 0을 표시하는 방식
  - 코드

    ex)

    oh_encoder = OneHotEncoder( )

    oh_encoder.fit(items)

    oh_labels = oh_encoder.transform(items)
  - get_dummies(데이터프레임명)
    - 원-핫 인코딩을 더 쉽게 지원
    - 문자열 카테고리 값을 숫자 형으로 변환할 필요 없이 바로 변환 가능
- 피처 스케일링과 정규화
  - 서로 다른 변수의 값 범위를 일정한 수준으로 맞추는 작업

- 표준화(Standardization)
  - 데이터의 피처 각각이 평균이 0이고 분산이 1인 가우시안 정규 분포를 가진 값으로 변환
  - 개별 데이터의 크기를 모두 똑같은 단위로 변경

$$x_i new = \frac{x_i - mean(x)}{stdev(x)}$$

- 정규화(Normalization)
  - 개별 데이터의 크기를 모두 똑같은 단위로 변경

$$x_i new = \frac{x_i - min(x)}{max(x) - min(x)}$$

  - 사이킷런의 Normalizer 모듈
    - 개별 벡터를 모든 피처 벡터의 크기로 나누어줌

$$x_i new = \frac{x_i}{\sqrt{x_i^2 + y_i^2 + z_i^2}}$$

- **StandardScaler**
  - 개별 피처를 평균이 0이고 분산이 1인 값으로 변환
  - 서포트 벡터 머신, 선형 회귀, 로지스틱 회귀는 데이터가 가우시안 분포를 가지고 있다고 가정하고 구현되었기 때문에 사전에 표준화를 적용하는 것은 예측 성능 향상에 중요한 요소가 됨
  - 코드

```
ex)
scaler = StandardScaler( )
scaler.fit(iris_df)
iris_scaled = scaler.transform(iris_df)
```

- **MinMaxScaler**
  - 데이터값을 0과 1 사이의 범위 값으로 변환 ** 음수 값이 있으면 -1에서 1값으로 변환
  - 코드

```
ex)
scaler = MinMaxScaler( )
scaler.fit(iris_df)
iris_scaled = scaler.transform(iris_df)
```

- 학습 데이터와 테스트 데이터의 스케일링 변환시 유의점
  - 가능하다면 전체 데이터의 스케일링 변환을 적용한 뒤 학습/테스트 데이터로 분리
  - 테스트 데이터 변환시 fit( )이나 fit_transform( )을 적용하지 않고 학습 데이터로 이미 fit( )된 Scaler 객체를 이용해 transform( )으로 변환

- 분류의 성능 평가 지표
  - 정확도(Accuracy)
  - 오차행렬(Confusion Matrix)
  - 정밀도(Precision)
  - 재현율(Recall)
  - F1 스코어
  - ROC AUC

01 정확도(Accuracy)

- 정확도
  - 실제 데이터에서 예측 데이터가 얼마나 같은지를 판단하는 지표

    ex) accuracy = accuracy_score(실제결과, 예측 결과)

    $$Accuracy = \frac{\text{예측 결과가 동일한 데이터 건수}}{\text{전체 예측 데이터 건수}}$$
  - 이진 분류의 경우 데이터의 구성에 따라 ML 모델의 성능을 왜곡할 수 있음
  - 불균형한 레이블 값 분포에서 적합한 평가 지표 X
  - 정확도 = (TN + TP) / (TN + FP + FN + TP)

    → Negative에 대한 예측 정확도만으로도 분류의 정확도가 높게 나오는 수치적인 판단 오류 발생

02 오차 행렬

- 오차 행렬
  - 학습된 분류 모델이 예측을 수행하면서 얼마나 헷갈리고(confused) 있는지 함께 보여주는 지표
  - 이진 분류에서 성능 지표로 잘 활용됨
  - 이진 분류의 예측 오류가 얼마인지 + 어떤 유형의 예측 오류가 발생하고 있는지 나타냄
  - 오차행렬의 4분명
    - TN : 예측값을 negative 값 0으로 예측했고 실제 값 역시 negative 값 0
    - FP : 예측값을 positive 값 1으로 예측했는데 반면 실제 값은 negative 값 0
    - FN : 예측값을 negative 값 0으로 예측했는데 반면 실제 값은 positive 값 1
    - TP : 예측값을 positive 값 1으로 예측했고 실제 값 역시 positive 값 1

    ex) confusion_matrix(실제 결과, 예측 결과)

03 정밀도와 재현율

- 정밀도(양성 예측도)
  - 예측을 Positive로 한 대상 중에 예측과 실제 값이 Positive로 일치한 데이터의 비율
  - 정밀도 = TP / (FP + TP)
  - Positive 예측 성능을 더욱 정밀하게 측정하기 위한 평가 지표
  - 실제 Negative 양성 데이터를 Positive로 잘못 예측하면 업무상 큰 영향이 발생하는 경우 중요 지표로 사용됨
  - TP↑ FP↓

    ex) precision = precision_score(실제결과, 예측 결과)
- 재현율
  - 예측을 Positive로 한 대상 중에 예측과 실제 값이 Positive로 일치한 데이터의 비율
  - 재현율 = TP / (FN + TP)
  - 민감도(Sensitivity), TPR(True Positive Rate)라고도 불림
  - 실제 Positive 양성 데이터를 Negative로 잘못 예측하면 업무상 큰 영향이 발생하는 경우 중요 지표로 사용됨

    ex) 암 판단 모델
  - TP↑ FN↓

    ex) recall = recall_score(실제결과, 예측 결과)
- 정밀도/재현율 트레이드오프(Trade-off)
  - 정밀도와 재현율은 상호 보완적인 평가 지표이므로 어느 한쪽을 강제로 높이면 다른 하나의 수치는 떨어짐
  - 분류 알고리즘
    - 예측 확률이 더 큰 레이블 값으로 예측
  - predict_proba( )
    - 개별 데이터 별로 예측 확률 결과 반환

      ex) lr_clf.predict_proba(X_test)
  - Binarizier 클래스

- 입력된 ndarray의 값을 지정된 threshold보다 같거나 작으면 0값으로, 크면 1값으로 변환해 반환
  ex) binarizer = Binarizer(threshold = 1.1)
  ninarizer.fit_transform(X)
- 임계값↓ → Positive로 예측할 확률↑ 재현율↑
- precision_recall_curve( )
  - 입력 파라미터
    : 실제 클래스 값 배열
    : Positive 열의 예측 확률 배열
    ex) precision_recall_curve(y_true, probas_pred)
  - 반환 값
    : 정밀도 - 임계값별 정밀도 값을 배열로 반환
    : 재현율 - 임계값별 재현율 값을 배열로 반환

● 정밀도와 재현율의 맹점
- 임계값의 변경은 두 수치를 상호 보완할 수 있는 수준에서 적용되어야 함
- 정밀도/재현율 수치 중 하나를 극단적으로 높이는 방법은 숫자 놀음에 불과함
- 정밀도/재현율 중 하나만 강조하는 상황 X

04 F1 스코어

● **F1** 스코어
- 정밀도와 재현율을 결합한 지표로 두 수치가 어느 한쪽으로 치우치지 않았을 때 상대적으로 높은 값 가짐

$$F1 = \frac{2}{\frac{1}{recall}+\frac{1}{precision}} = 2 * \frac{precision*recall}{precision+recall}$$

- f1_score( )
  ex) f1 = f1_score(y_test, pred)

05 ROC 곡선과 AUC

● **ROC** 곡선
- Receiver Operation Characteristic Curve
- 수신자 판단 곡선
- FPR(False Positive Rate)이 변할 때 TPR(True Positive Rate)이 어떻게 변하는지 나타내는 곡선
  - FPR을 X축, TPR을 Y축으로
- TPR(재현율, 민감도)
  - 실제값 Positive(양성)가 정확히 예측돼야 하는 수준
  - TP / (FN + TP)
- 특이성(TNR)(True Negative Rate)
  - 실제값 Negative가 정확히 예측돼야 하는 수준
  - TN / (FP + TN)
- FPR
  - FP / (FP + TN)
  - 1 - TNR
  - 1 - TNR(특이성)
- roc_curve( )
  - 입력 파라미터
    : 실제 클래스 값 배열
    : predict_proba( )의 반환 값 array에서 Positive 열의 예측 확률이 보통 사용됨
    ex) roc_curve(y_test, pred_proba_class1)
  - 반환 값
    - fpr
      : fpr 값을 array로 반환
    - tpr
      : tpr 값을 array로 반환
    - thresholds
      : thresold 값 array

● **AUC(Area Under Curve)**
- ROC 곡선에 기반한 AUC 스코어는 이진 분류의 예측 성능 측정에서 중요하게 사용되는 지표임
- ROC 곡선 밑의 면적을 구한 것
- 일반적으로 1에 가까울수록 좋은 수치
- AUC 수치가 커지기 위해 FPR이 작은 상태에서 얼마나 큰 TPR을 얻을 수 있는지가 관건