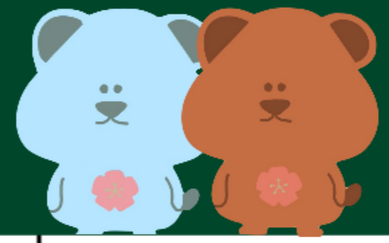
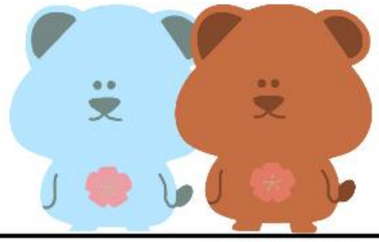


1주차 발표

DA팀 이다현, 최하경

EWHA EURON

목차



EWHA EURON

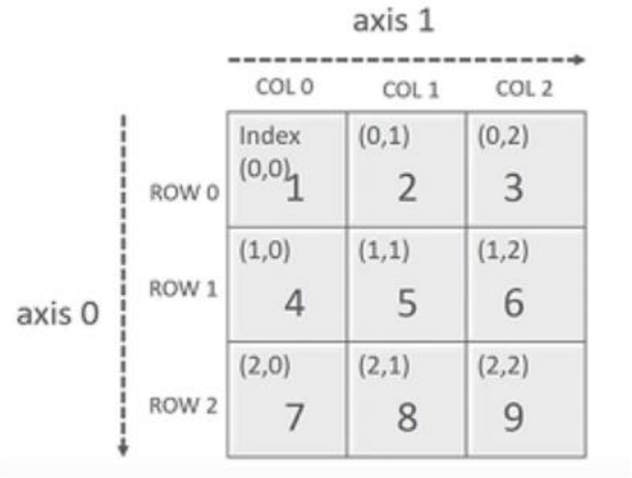
01. 파이썬 기반의 머신러닝과 생태계 이해

EWHA EURON

1.3 넘파이

📌 넘파이 모듈 불러오기

```
import numpy as np
```



📌 넘파이 함수 및 메서드 정리

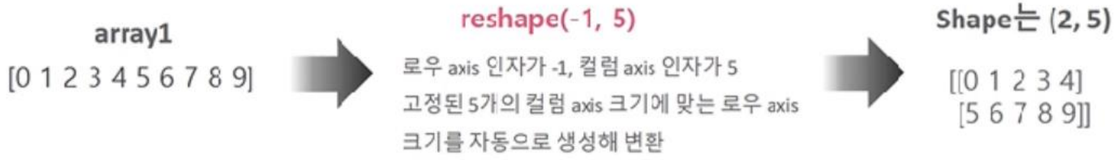
① ndarray 배열 생성

Code	설명	예시
np.array( )	인자를 받아 ndarray로 변환	
np.arange( )	0부터 (인자-1)까지 1차원 ndarray 생성	
np.zeros( )	0으로 입력받은 shape만큼의 ndarray 생성	
np.ones( )	1로 입력받은 shape만큼의 ndarray 생성	

1.3 넘파이

② ndarray 속성 확인

Code	설명	예시
ndarray.shape	ndarray의 차원(행,열) 출력	
ndarray.ndim	ndarray의 차원(행만) 출력	
ndarray.dtype	ndarray의 데이터 타입(int32,bool,etc) 출력	
ndarray.reshape( )	1로 입력받은 shape의 ndarray 생성	



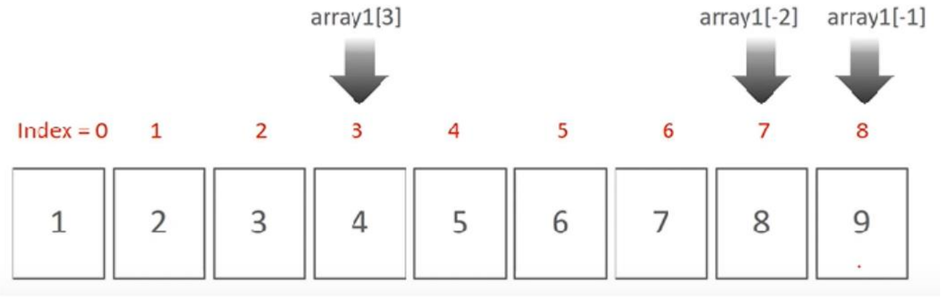
1.3 넘파이

📌 넘파이 데이터 세트 선택하기 – 인덱싱 (Indexing)

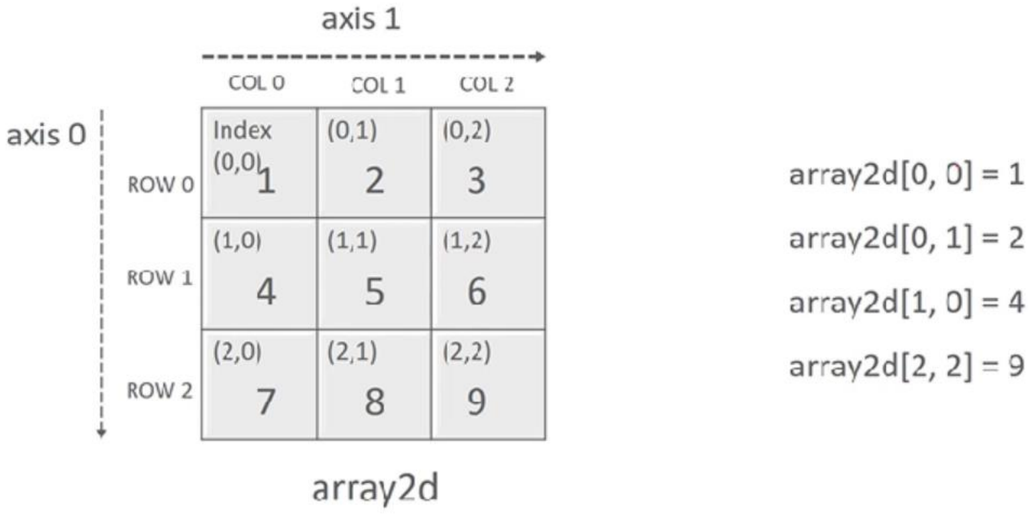
① 특정한 데이터만 추출

- 원하는 위치의 인덱스 값을 지정해 해당 위치의 데이터를 반환

(i) 1차원



(ii) 2차원



array2d[0, 0] = 1

array2d[0, 1] = 2

array2d[1, 0] = 4

array2d[2, 2] = 9

1.3 넘파이

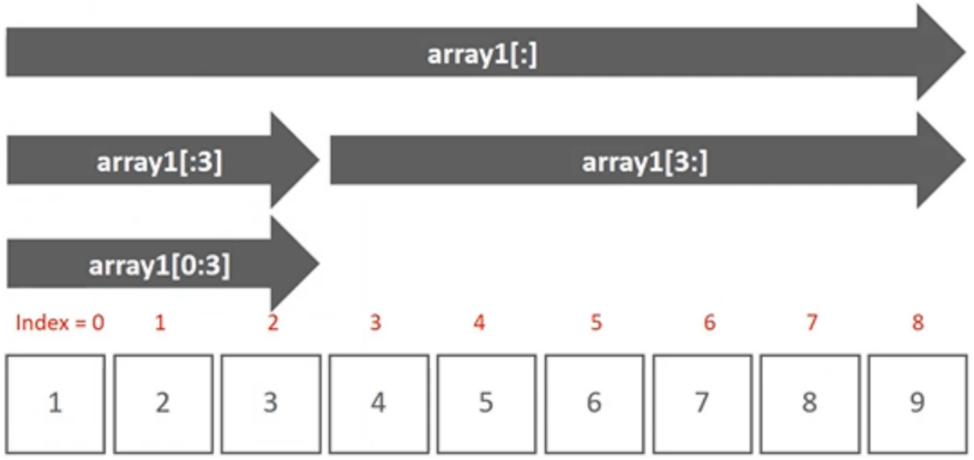
📌 넘파이 데이터 세트 선택하기 – 인덱싱 (Indexing)

② 슬라이싱 (Slicing)

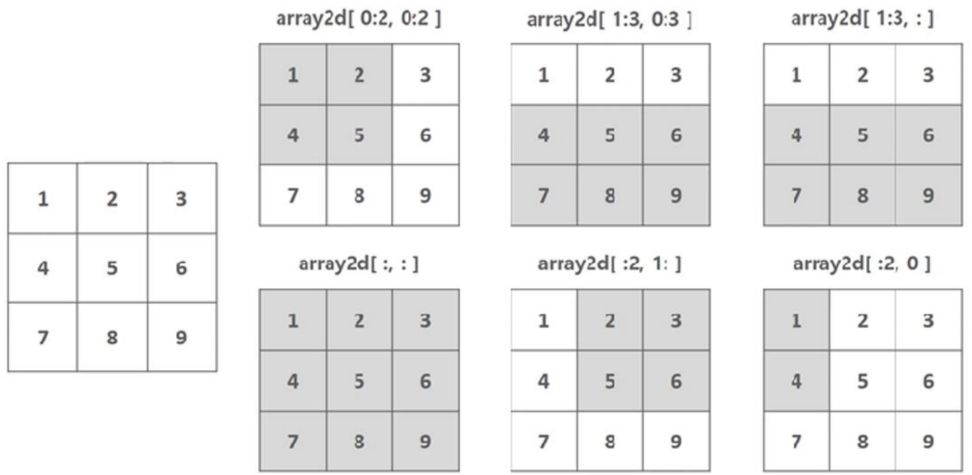
- ':' 을 이용해 연속된 인덱스상의 ndarray를 추출하는 방식

★ a:b -> a부터 (b-1)까지

(i) 1차원



(ii) 2차원



1.3 넘파이

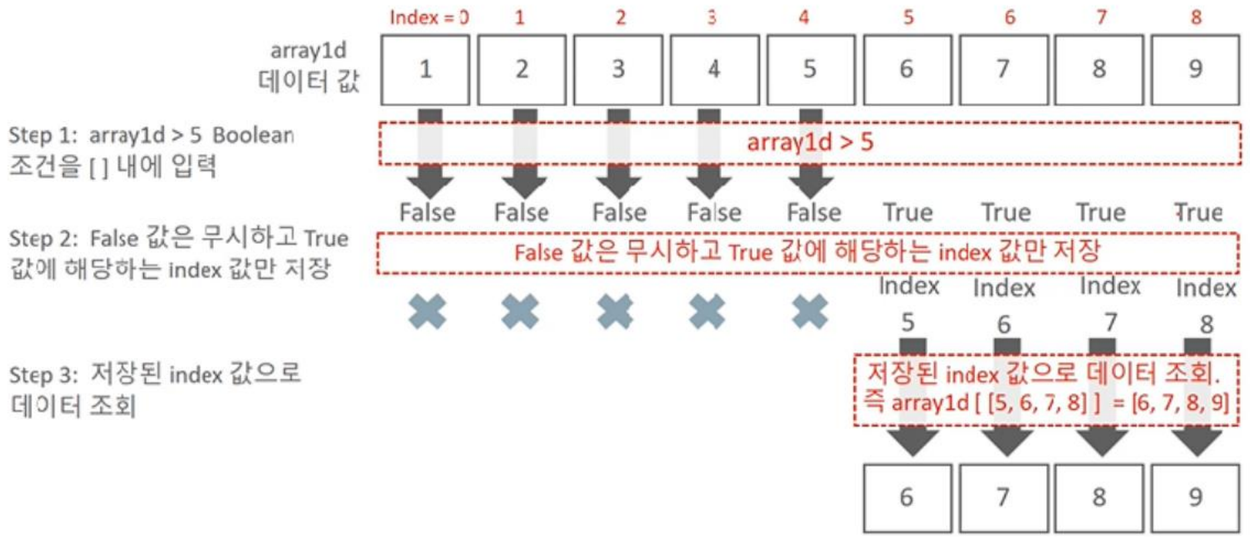
📌 넘파이 데이터 세트 선택하기 – 인덱싱 (Indexing)

③ 팬시 인덱싱 (Fancy Indexing)

- 특정 조건에 따라 T/F 값 인덱싱 집합을 기반으로 True에 해당하는 데이터의 ndarray 반환

- 매커니즘

```
array1d[array1>5]
```



1.3 넘파이

📌 행렬의 정렬

★ 기본적으로 오름차순으로 행렬 내 원소 정렬
★ 내림차순 정렬 : [::-1] 적용

① np.sort( )

- 넘파이에서 sort( )를 호출하는 방식
- 원 행렬을 그대로 유지한 채 원 행렬의 정렬된 행렬을 반환 => 저장 필요

② ndarray.sort( )

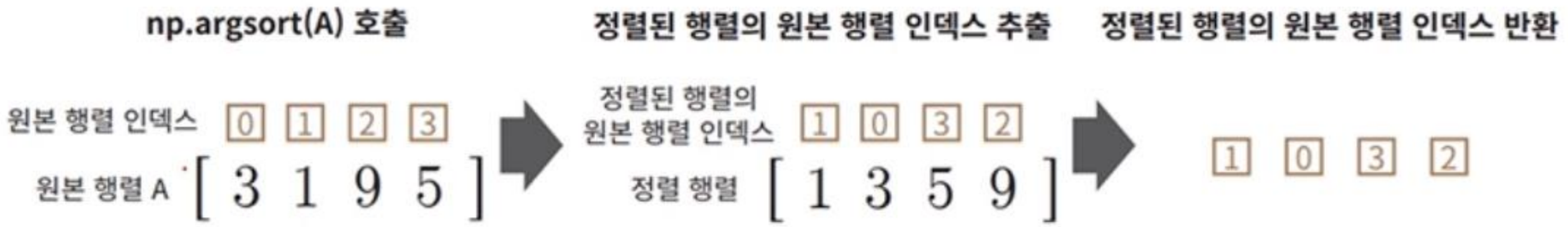
- 행렬 자체에서 sort( )를 호출하는 방식
- 원 행렬 자체를 정렬한 상태로 변환 (반환값은 None) => 저장할 필요 X

1.3 넘파이

📌 행렬의 정렬

③ np.argsort( )

- 정렬 행렬의 원본 행렬 인덱스를 ndarray 형으로 변환



1.3 넘파이

📌 선형대수 연산 – 행렬 내적과 전치 행렬 구하기

① np.dot( )

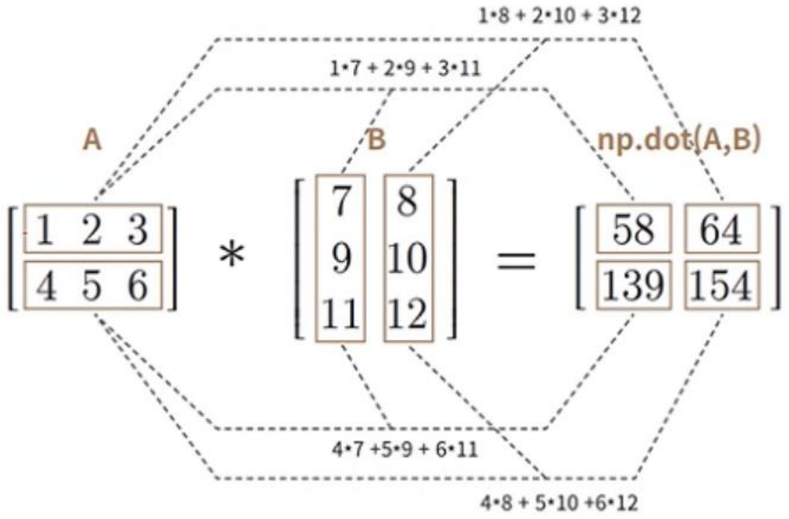
- 행렬 내적 (행렬 곱)
- 왼쪽 행렬의 열 개수와 오른쪽 행렬의 행 개수가 동일해야 함 (A[i,j]*B[j,k])

$$C = AB$$

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & \ddots & \cdots & \vdots \\ \vdots & \cdots & \cdots & \vdots \\ a_{m1} & \cdots & \cdots & a_{mn} \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1r} \\ b_{21} & \ddots & \cdots & \vdots \\ \vdots & \cdots & \cdots & \vdots \\ b_{n1} & \cdots & \cdots & b_{nr} \end{bmatrix}$$

$$c_{ij} = a_{i1}b_{1j} + a_{i2}b_{2j} + \cdots + a_{in}b_{nj} = \sum_{k=1}^{n} a_{ik}b_{kj}$$

$$\underset{(m \times n)}{A} \quad \underset{(n \times r)}{B} = \underset{(m \times r)}{C}$$

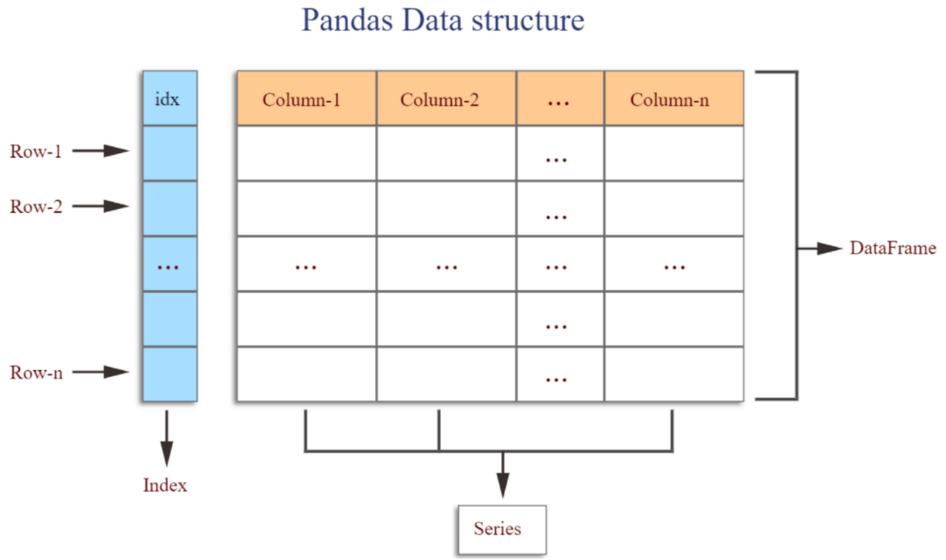


1.4 판다스

📌기본 이해

① Pandas의 핵심 개체 : DataFrame

② Series와 DataFrame



- Series : 칼럼이 하나뿐인 데이터 구조체
- DataFrame : 칼럼이 여러 개인 데이터 구조체 (여러 개의 Series)

1.4 판다스

📌 판다스 시작 – 파일을 DataFrame으로 로딩 ,기본 API

① 판다스 모듈 임포트

```
import pandas as pd
```

② DataFrame 불러오기

Code	설명	예시
pd.read_csv( )	Csv 파일 포맷 변환을 위한 API 디폴트 필드 구분문자 : ‘,’ Sep 인자를 활용 (sep= ‘\t’)	
pd.read_table( )	디폴트 필드 구분문자 : ‘\t '	
pd.read_fwf( )	고정 길이 기반의 칼럼 포맷 대상 (잘 사용하지 않음)	

1.4 판다스

📌 판다스 시작 – 파일을 DataFrame으로 로딩 ,기본 API

③ DataFrame 정보 확인하기

Code	설명	예시
DataFrame.head( )	DataFrame의 맨 앞에 있는 N개의 row 반환 (default : 5)	
DataFrame.shape	DataFrame의 행과 열을 튜플 형태로 반환	
DataFrame.info ()	총 데이터 건수와 칼럼별 데이터 타입, Null 건수	
DataFrame.describe( )	숫자형 칼럼에 대한 개략적인 데이터 분포도 (non-null 건수, mean, std, min, max, quantile)	
DataFrame['열'].value_counts( )	해당 칼럼값의 유형과 건수 확인	

1.4 판다스

📌 DataFrame의 칼럼 데이터 세트 생성과 수정

① [ ] 연산자 이용

- DataFrame['열']=숫자
- DataFrame['열']=수식

1.4 판다스

📌DataFrame 데이터 삭제

① DataFrame.drop()

- labels : drop할 칼럼이나 행 지정 ([ ] 안에 넣어서 입력)
- axis : 특정 칼럼 또는 특정 행 drop (axis=0 : row, axis=1 : column)
- inplace : drop한 데이터프레임을 원본으로 저장 (default : False)

  : inplace=True 일 경우 반환값 None => 따로 저장할 필요 X

1.4 판다스

📌 Index 객체

① DataFrame.index

- DataFrame의 인덱스 객체 추출
- **출력값 :** `RangeIndex(start=0, stop=891, step=1)`

② DataFrame.index.values

- DataFrame의 인덱스 객체를 실제 값 array로 변환
- **출력값 :**

```
[  0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17
  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35
  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53
  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71
```

1.4 판다스

📌 Index 객체

① DataFrame.index

- DataFrame의 인덱스 객체 추출
- **출력값 :** `RangeIndex(start=0, stop=891, step=1)`

② DataFrame.index.values

- DataFrame의 인덱스 객체를 1차원 array로 변환 (단일 값 변환 / 슬라이싱 가능)
- **출력값 :**

```
[  0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17
  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35
  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53
  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71
```

1.4 판다스

📌 Index 객체

① DataFrame.reset_index( )

- DataFrame 인덱스를 새롭게 연속 숫자형 (0~)으로 할당
- 기존 인덱스는 'index' 라는 새로운 칼럼명으로 추가
- 주요 파라미터

  (i) drop : 기존 인덱스를 새로운 칼럼으로 추가하지 않고 삭제 (default=False)

  (ii) inplace : reset_index한 데이터프레임을 원본으로 저장 (default=False)

1.4 판다스

📌 데이터 셀렉션 및 필터링

① [ ] 연산자

- DataFrame의 칼럼만 지정하는 **'칼럼 지정 연산자'**

  슬라이싱을 이용해 인덱스 기반으로도 추출 가능
  but 사용 지양

- 단일 칼럼 추출

```
titanic_df[ 'Pclass' ]
```

- 여러 칼럼 추출 ([ ] 안에 해당 칼럼들이 포함된 [ ] 입력)

```
titanic_df[ ['Survived', 'Pclass'] ]
```

- 불린 인덱싱 가능

```
titanic_df[ titanic_df['Pclass'] == 3]
```

EWHA EURON

1.4 판다스

📌 데이터 셀렉션 및 필터링

② DataFrame.iloc[ ]

- 위치 기반 인덱싱
- 행과 열 값으로 integer 혹은 integer형의 슬라이싱, 팬시 인덱싱만 가능
- 칼럼 명칭을 입력하면 오류 발생

```
data_df.iloc[0, 0]
```

```
data_df.iloc[0, 'Name']
```

1.4 판다스

📌 데이터 셀렉션 및 필터링

③ DataFrame.loc[ ]

- 명칭 기반 인덱싱
- 인덱스는 숫자형으로 입력 가능!!
- 예외적으로 슬라이싱이 시작값~종료값의 범위를 가짐.

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

```
'Chulmin'
```

```
data_df_reset.loc[1, 'Name']
```

```
'Chulmin'
```

1.4 판다스

📌 데이터 셀렉션 및 필터링

④ 불린 인덱싱

- [ ] 연산자와 DataFrame.loc[ ] 사용
- 복합 조건 결합해 사용 가능 (& , | , ~)
- [ ] 연산자의 경우 : DatFrame[[조건][열 이름들]]

```
titanic_df[titanic_df['Age'] > 60][['Name','Age']]
```

- .loc[ ]의 경우 : DataFrame.loc[[조건],[열 이름들]]

```
titanic_df.loc[titanic_df['Age'] > 60, ['Name','Age']]
```

1.4 판다스

📌 정렬, Aggregation 함수, GroupBy 적용

① DataFrame.sort_values( )

- DataFrame과 Series 정렬
- 주요 파라미터

  (i) by : 정렬의 기준이 될 열(들)을 리스트형으로 입력

  (ii) ascending : 오름차순으로 정렬 여부 (default : True)

  (iii) inplace : 정렬한 데이터프레임을 원본으로 저장 (default : false)

```
titanic_sorted = titanic_df.sort_values(by=['Pclass', 'Name'], ascending=False)
```

1.4 판다스

📌 정렬, Aggregation 함수, GroupBy 적용

② Aggregation 함수

- 기본 연산을 도와주는 함수들의 집합 (min( ), max( ), sum( ), median( ), count( ))
- DataFrame 전체 혹은 특정 칼럼들에 해당 aggregation 적용
- 전체 칼럼들 : 데이터프레임 뒤에 함수 적용

```
titanic_df.count()
```

- 특정 칼럼들 : 데이터프레임[[칼럼들]] 뒤에 함수 적용

```
titanic_df[['Age', 'Fare']].mean()
```

- axis 설정에 따라 적용되는 방향 달라짐 (axis=0 : 행 / axis=1 : 열)

1.4 판다스

📌 정렬, Aggregation 함수, GroupBy 적용

③ DataFrame.groupby( )

- 입력 파라미터 by에 칼럼을 입력하면 대상 칼럼으로 groupby
- 반환된 DataFrame Groupby 객체에 aggregation 함수를 호출

  => groupby( ) 대상 칼럼을 제외한 모든 칼럼에 해당 aggregation 함수 적용

  => 특정 칼럼들에만 적용 : .groupby( )[열].함수( ) / .groupby( )[[열들]].함수( )
- .agg( )을 이용해 여러 개의 aggregation 함수 적용

  => 하나의 칼럼에 여러 개의 함수 : .agg([함수들])

  => 칼럼에 따라 서로 다른 함수 : .agg(dic( ))

```
titanic_df.groupby('Pclass')['Age'].agg([max, min])
```

```
agg_format={'Age':'max', 'SibSp':'sum', 'Fare':'mean'}
titanic_df.groupby('Pclass').agg(agg_format)
```

1.4 판다스

📌 결손 데이터 처리하기

① DataFrame.isna( )

- 모든 칼럼 값이 NaN인지 아닌지를 True/False로 반환
- 결손 데이터 개수를 확인하기 위해서는 DataFrame.isna( ).sum( )

② DataFrame.fillna( )

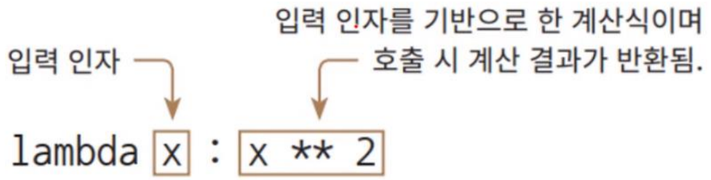
- 결손 데이터를 다른 값으로 대체 가능
- 특정 칼럼에 대해서만 적용 : DataFrame[열].fillna( ) / DataFrame[[열]].fillna( )
- inplace=True 혹은 다른 이름으로 저장해야 fillna( ) 실행 값 저장됨

1.4 판다스

📌 apply lambda 식으로 데이터 가공

① lambda

- 함수의 선언과 함수 내의 처리를 한 줄로 변환한 식



② DataFrame.apply(lambda 식)

- 특정 칼럼에 대해서만 적용 : DataFrame[열]. apply( ) / DataFrame[[열]].apply( )
- if else문 : list comprehension 사용
- 너무 길어질 경우 따로 함수 정의하는 것도 방법

02. 사이킷런으로 시작하는 머신러닝

EWHA EURON

2.2 첫 번째 머신러닝 만들어보기

📌 프로세스 정리

① **데이터 세트 분리** : 데이터를 학습 데이터와 테스트 데이터로 분리

② **모델 학습** : 학습 데이터 기반으로 ML 알고리즘 적용해 모델 학습

③ **예측 수행** : 학습된 ML 모델을 이용해 테스트 데이터 예측

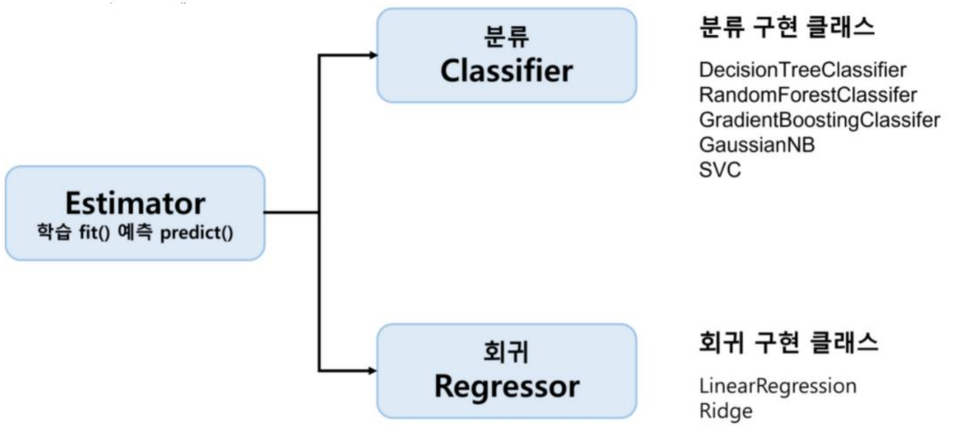
④ **평가** : 예측된 결과값과 테스트 데이터의 실제 결과값 비교해 모델 성능 평가

2.3 사이킷런의 기반 프레임워크 익히기

📌Estimator 이해 및 fit( ), predict( ) 메서드

① 지도학습

- 분류(Classificaiton)과 회귀(Regression)로 구성 -> Estimator 클래스
- fit( )과 predict( ) 내부 구현
- Evaluation 함수, 하이퍼파라미터 튜닝 클래스의 경우 이를 인자로 받음



2.3 사이킷런의 기반 프레임워크 익히기

📌Estimator 이해 및 fit( ), predict( ) 메서드

② 비지도학습

- 차원 축소, 클러스터링, 피처 추출(Feature Extraction) 등의 클래스
- fit( )과 transform( ) 내부 구현 (하나로 결합한 fit_transform( ) 제공)

(i) fit( ) : 입력 데이터의 형태에 맞춰 데이터를 변환하기 위한 사전 구조 맞춤

(ii) transform( ) : 이후 입력 데이터의 차원 변환, 클러스터링 등 실제 작업 수행

2.3 사이킷런의 기반 프레임워크 익히기

📌 사이킷런의 주요 모듈

① 피처 처리

모듈명	설명	예시
Sklearn.preprocessing	데이터 전처리에 필요한 기능	인코딩, 정규화, 스케일링
Sklearn.feature_selection	알고리즘에 큰 영향을 미치는 피처를 우선순위대로 셀렉션 작업 수행	-
Sklearn.feature_extraction	텍스트 데이터나 이미지 데이터의 벡터화된 피처 추출	Count Vectorizer, Tf-idf Vectorizer 등

2.3 사이킷런의 기반 프레임워크 익히기

📌 사이킷런의 주요 모듈

② 피처 처리 & 차원 축소

모듈명	설명	예시
Sklearn.decomposition	차원 축소 관련된 알고리즘	PCA, NMF, Truncated SVD

③ 데이터 분리, 검증 & 파라미터 튜닝

모듈명	설명	예시
Sklearn.model_selection	교차 검증을 위한 학습/테스트 분리, 최적 파라미터 추출	train_test_split, GridSearchCV 등

2.3 사이킷런의 기반 프레임워크 익히기

📌 사이킷런의 주요 모듈

④ 평가

모듈명	설명	예시
Sklearn.metrics	성능 측정 방법 제공	Accuracy, Precision, Recall, ROC-AUC, RMSE 등

2.3 사이킷런의 기반 프레임워크 익히기

📌 사이킷런의 주요 모듈

⑤ ML 알고리즘

모듈명	설명	예시
Sklearn.ensemble	앙상블 알고리즘 제공	RandomForest, Adaboost, gradient boosting 등
Sklearn.linear_model	회귀 알고리즘 제공	선형회귀, 릿지, 라쏘, 로지스틱 등
Sklearn.naïve_bayes	나이브 베이즈 알고리즘 제공	가우시안 NB, 다항분포 NB 등

2.3 사이킷런의 기반 프레임워크 익히기

📌 사이킷런의 주요 모듈

⑤ ML 알고리즘

모듈명	설명	예시
Sklearn.neighbors	최근접 이웃 알고리즘 제공	KNN 등
Sklearn.svm	서포트 벡터 머신 알고리즘 제공	-
Sklearn.tree	의사 결정 트리 알고리즘 제공	Decision tree 등
Sklearn.cluster	비지도 클러스터링 알고리즘 제공	K-평균, 계층형, DBSCAN

2.4 Model Selection 모듈 소개

📌 학습/테스트 데이터 세트 분리 – train_test_split( )

① train_test_split( )

```
from sklearn.model_selection import train_test_split
```

- 학습/테스트 데이터 세트 분리
- 반환값 : train_X, test_X, train_y, test_y 튜플 형태
- 주요 파라미터

(i) test_size : 전체에서 테스트 데이터 세트 크기를 얼마로 샘플링할 것인가 (default : 0.25)

(ii) shuffle : 데이터를 분리하기 전 분산시켜 효율적인 학습/테스트 데이터 분리 (default : True)

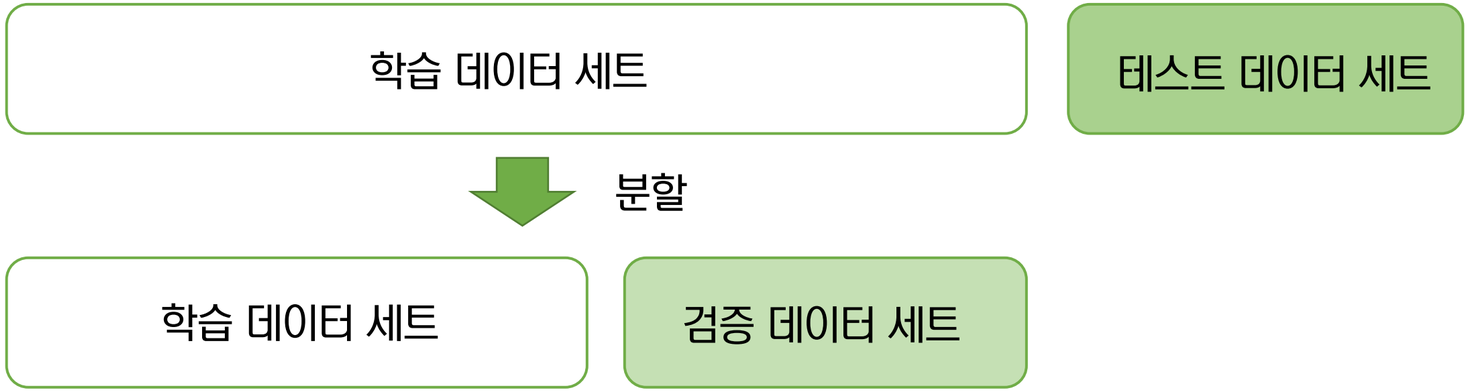
(iii) random_state : 호출할 때마다 동일한 학습/테스트 데이터 세트를 생성하기 위해 주어지는 난수 값

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

2.4 Model Selection 모듈 소개

📌 교차 검증

- 과적합(Overfitting) 방지
- 데이터 편중을 막기 위해 별도의 여러 세트로 구성된 학습/테스트 데이터 세트에서 학습과 평가 수행



2.4 Model Selection 모듈 소개

📌 교차 검증

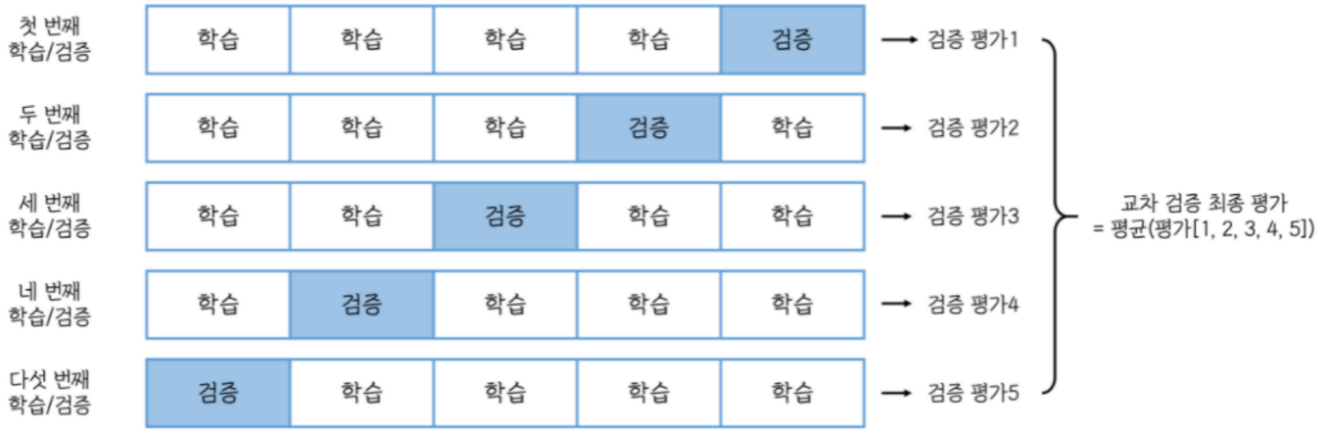
① K 폴드 교차 검증

```
from sklearn.model_selection import KFold
```

- K개의 데이터 폴드 세트를 만들어 K번만큼 각 폴드 세트에 학습과 검증 평가를 반복적으로 수행
- n_splits 파라미터를 이용해 데이터 세트 지정

★ Kfold.split( )
폴드별 학습용, 검증용 데이터의 인덱스를 array로 반환

```
kfold = KFold(n_splits=5)
```



2.4 Model Selection 모듈 소개

📌 교차 검증

② Stratified K 폴드

```
from sklearn.model_selection import StratifiedKFold
```

- 이산값 형태의 레이블(Y)을 가진 데이터에 한해 적용 – 분류, 로지스틱 회귀 등
- ★불균형한 (imbalanced) 분포도를 가진 Y 데이터 집합을 위한 K 폴드 방식
- 원본 데이터의 Y 분포를 먼저 고려한 뒤 이 분포와 동일하게 학습/검증 데이터 분배
- 코드 예 :

```
skfold = StratifiedKFold(n_splits=3)
```

★ 불균형한 분포도

특정 Y 값이 특이하게 많거나 매우 적어서

값의 분포가 한쪽으로 치우치는 것

2.4 Model Selection 모듈 소개

📌 교차 검증

② Stratified K 폴드

```
from sklearn.model_selection import StratifiedKFold
```

- 코드 예시

```
# StratifiedKFold의 split( ) 호출시 반드시 레이블 데이터 셋도 추가 입력 필요
for train_index, test_index  in skfold.split(features, label):
    # split( )으로 반환된 인덱스를 이용하여 학습용, 검증용 테스트 데이터 추출
    X_train, X_test = features[train_index], features[test_index]
    y_train, y_test = label[train_index], label[test_index]
    #학습 및 예측
    dt_clf.fit(X_train , y_train)
    pred = dt_clf.predict(X_test)

    # 반복 시 마다 정확도 측정
    n_iter += 1
    accuracy = np.round(accuracy_score(y_test,pred), 4)
    train_size = X_train.shape[0]
    test_size = X_test.shape[0]
    print('₩n#{0} 교차 검증 정확도 :{1}, 학습 데이터 크기: {2}, 검증 데이터 크기: {3}'
          .format(n_iter, accuracy, train_size, test_size))
    print('#{0} 검증 세트 인덱스:{1}'.format(n_iter,test_index))
    cv_accuracy.append(accuracy)

# 교차 검증별 정확도 및 평균 정확도 계산
print('₩n## 교차 검증별 정확도:', np.round(cv_accuracy, 4))
print('## 평균 검증 정확도:', np.mean(cv_accuracy))
```

★ 교차 검증 과정★

❶ 폴드 세트 설정

❷ for 루프에서 반복적으로 학습/검증용 인덱스 추출

❸ 학습과 예측 수행해 예측 성능 반환

EWHA EURON

2.4 Model Selection 모듈 소개

📌 교차 검증

③ cross_val_score( )

```
from sklearn.model_selection import cross_val_score , cross_validate
```

- 교차 검증 과정을 한꺼번에 수행해주는 API
- 반환값 : 배열 형태의 지정된 성능 지표 측정값
- 주요 파라미터

  (i) estimator : Classifier 또는 Regressor

  (ii) X : 피처 데이터 세트 (X에 해당하는 데이터)

  (iii) y : 레이블 데이터 세트 (target 값)

  (iv) scoring : 예측 성능 평가 지표

  (v) cv : 교차 검증 폴드 수 (default : 5)

2.4 Model Selection 모듈 소개

📌 교차 검증

https://docs.w3cub.com/scikit_learn/modules/generated/sklearn.model_selection.cross_validate

④ cross_validate( )

```
from sklearn.model_selection import cross_val_score , cross_validate
```

- 교차 검증 과정을 한꺼번에 수행해주는 API
- 반환값 : dict 형태의 각 폴드별 test_score, fit_time 등 반환
- 주요 파라미터

  (i) estimator : Classifier 또는 Regressor

  (ii) X : 피처 데이터 세트 (X에 해당하는 데이터)

  (iii) y : 레이블 데이터 세트 (target 값)

  (iv) scoring : 예측 성능 평가 지표 – [‘accuracy’ , ‘roc_auc’] 처럼 리스트형으로 여러 개 지정 가능

  (v) cv : 교차 검증 폴드 수 (default : 3)

  (vi) return_train_score : 훈련 폴드에 대한 점수(train_score)와 score_time (default : True)

2.4 Model Selection 모듈 소개

📌 GridSearchCV – 교차검증과 최적 하이퍼 파라미터 튜닝을 한 번에

① 기본 정보

- 교차 검증을 기반으로 하이퍼 파라미터의 최적 값을 찾는 API
- 주요 파라미터

(i) estimator : Classifier, Regressor 또는 Pipeline

(ii) param_grid : (dict 형태) estimator 튜닝을 위해 파라미터명과 여러 파라미터 값들 지정

(iiv) scoring : 예측 성능 평가 지표

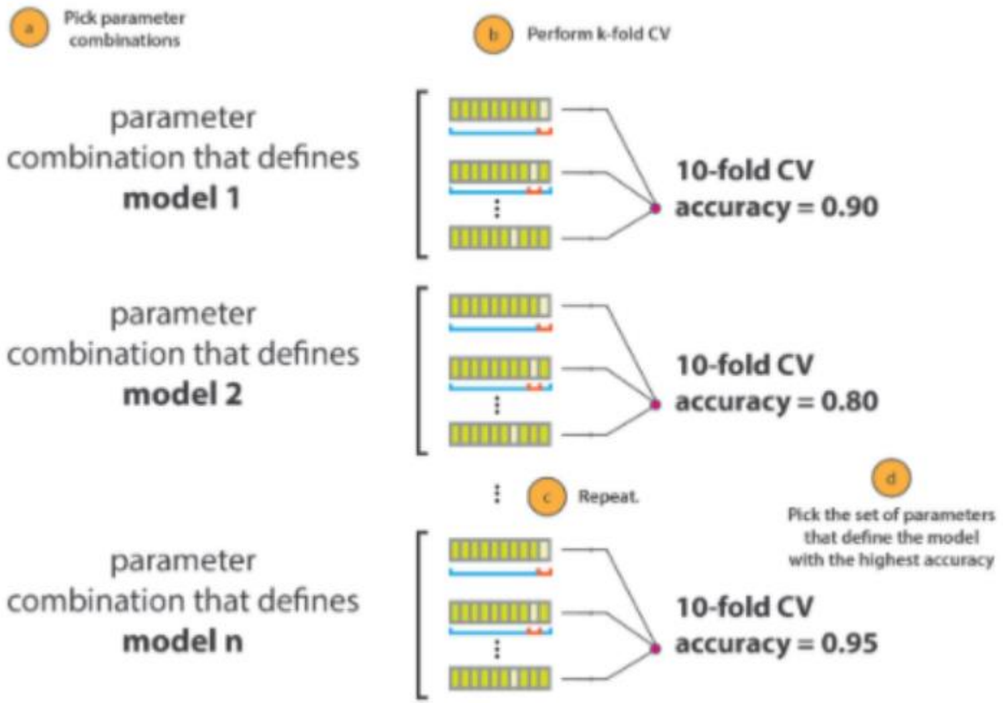
(iv) cv : 교차 검증을 위해 분할되는 학습/테스트 세트의 개수

(v) refit : 가장 최적의 하이퍼 파라미터를 찾은 뒤 estimator 객체에 재학습 (default : True)

2.4 Model Selection 모듈 소개

📌 GridSearchCV – 교차검증과 최적 하이퍼 파라미터 튜닝을 한 번에

② 과정



❶ 파라미터 조합들 중 하나를 선택해 모델 생성

❷ 입력받은 cv만큼 교차검증해 성능 평가

❸ 모든 파라미터 조합들에 대해 이 과정을 반복

❹ 가장 성능이 좋은 파라미터 조합을 채택

2.4 Model Selection 모듈 소개

📌 GridSearchCV – 교차검증과 최적 하이퍼 파라미터 튜닝을 한 번에

② 과정 – 코드 예시

```
# 데이터를 로딩하고 학습데이타와 테스트 데이터 분리
iris = load_iris()
X_train, X_test, y_train, y_test = train_test_split(iris_data.data, iris_data.target,
                                                    test_size=0.2, random_state=121)
dtree = DecisionTreeClassifier()

### parameter 들을 dictionary 형태로 설정
parameters = {'max_depth':[1,2,3], 'min_samples_split':[2,3]}
```

헷갈리지 말자!!
★ train_test_split은 train, test data 분리
★ 교차검증은 train data 내에서 시행

```
# param_grid의 하이퍼 파라미터들을 3개의 train, test set fold 로 나누어서 테스트 수행 설정.
### refit=True 가 default 임. True이면 가장 좋은 파라미터 설정으로 재 학습 시킴.
grid_dtree = GridSearchCV(dtree, param_grid=parameters, cv=3, refit=True)

# 붓꽃 Train 데이터로 param_grid의 하이퍼 파라미터들을 순차적으로 학습/평가 .
grid_dtree.fit(X_train, y_train)
```

⬅ 여기에서 앞서 말한 과정들 수행

2.4 Model Selection 모듈 소개

📌 GridSearchCV – 교차검증과 최적 하이퍼 파라미터 튜닝을 한 번에

③ attributes

Attribute	반환값	코드 예시
cv_results_	(dict 형태) 파라미터별 score	grid.cv_results_
best_estimators_	(estimator) 파라미터가 최적화된 모델	grid.best_estimators_
best_score_	(float) 가장 좋은 모델의 성능 평균값	grid.best_score_
best_params_	(dict) 최적의 파라미터	grid.best_params_

https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html

2.5 데이터 전처리

📌 결손값 (Null 값) 처리

① 피처들 중 Null 값이 적은 경우 : 해당 피처의 평균값 등으로 대체

: 이때 피처의 중요도에 따라 대체값 선정에 유의

② 피처들 중 Null 값이 많은 경우 : 해당 피처를 drop하는게 일반적

★ Null 값의 많고 적음의 기준 : 해당 피처에서의 Null 값 비율로 따지는 게 일반적

but 절대적인 기준은 없다!!

```
# Train missing values (in percent)
train_missing = (train.isnull().sum() / len(train)).sort_values(ascending = False)

# Test missing values (in percent)
test_missing = (test.isnull().sum() / len(test)).sort_values(ascending = False)


# Identify missing values above threshold
train_missing = train_missing.index[train_missing > 0.75]
test_missing = test_missing.index[test_missing > 0.75]
```

여기 예시에서는 75% 이상이 Null 값인 경우 drop

EWHA EURON

2.5 데이터 전처리

📌데이터 인코딩

① LabelEncoder

```
from sklearn.preprocessing import LabelEncoder
```

- 문자열 값을 숫자형 카테고리 값으로 변환
- 일괄적인 숫자 값으로 변환되면서 특정 ML 알고리즘에서는 예측성능 저하 문제 발생
- 트리 계열의 ML 알고리즘에서 사용 추천

```
items=['TV','냉장고','전자렌지','컴퓨터','선풍기','선풍기','믹서','믹서']

# LabelEncoder를 객체로 생성한 후 , fit( ) 과 transform( ) 으로 label 인코딩 수행.
encoder = LabelEncoder()
encoder.fit(items)
labels = encoder.transform(items)
print('인코딩 변환값:',labels)
```

2.5 데이터 전처리

📌데이터 인코딩

② One-Hot Encoding

```
from sklearn.preprocessing import OneHotEncoder
```

- 행 형태의 피처 고유 값을 열 형태로 차원 변환
- 고유 값에 해당하는 칼럼에만 1, 나머지는 0 표시
- 예시

```
# 먼저 숫자값으로 변환을 위해 LabelEncoder로 변환합니다.
encoder = LabelEncoder()
encoder.fit(items)
labels = encoder.transform(items)
# 2차원 데이터로 변환합니다.
labels = labels.reshape(-1,1)

# 원-핫 인코딩을 적용합니다.
oh_encoder = OneHotEncoder()
oh_encoder.fit(labels)
oh_labels = oh_encoder.transform(labels)
print('원-핫 인코딩 데이터')
print(oh_labels.toarray())
print('원-핫 인코딩 데이터 차원')
print(oh_labels.shape)
```

★ 변환 전 입력값으로 2차원 데이터 필요

원본 데이터

상품 분류
TV
냉장고
전자렌지
컴퓨터
선풍기
선풍기
믹서
믹서

원-핫 인코딩

상품분류_TV	상품분류_냉장고	상품분류_믹서	상품분류_선풍기	상품분류_전자렌지	상품분류_컴퓨터
1	0	0	0	0	0
0	1	0	0	0	0
0	0	0	0	1	0
0	0	0	0	0	1
0	0	0	1	0	0
0	0	0	1	0	0
0	0	1	0	0	0
0	0	1	0	0	0

2.5 데이터 전처리

📌데이터 인코딩

③ pd.get_dummies( )

- 판다스에서 제공하는 인코딩 방법
- 데이터프레임을 인수로 받아야 함

```
import pandas as pd

df = pd.DataFrame({'item':['TV','냉장고','전자렌지','컴퓨터','선풍기','선풍기','믹서','믹서'] })
pd.get_dummies(df)
```

2.5 데이터 전처리

📌 피처 스케일링

① 표준화

- 각 피처 값을 가우시안 정규 분포(평균=0, 분산=1)를 가진 값으로 변환
- StandardScaler 이용
- 특히 SVM, 선형회귀, 로지스틱 회귀 적용하기 전 꼭 수행!!!

```
from sklearn.preprocessing import StandardScaler

# StandardScaler객체 생성
scaler = StandardScaler()
# StandardScaler 로 데이터 셋 변환. fit( ) 과 transform( ) 호출.
scaler.fit(iris_df)
iris_scaled = scaler.transform(iris_df)
```

2.5 데이터 전처리

📌 피처 스케일링

② 정규화

$$x_{scaled} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

- 서로 다른 피처의 크기를 0과 1 사이 값으로 통일 (음수가 있을 경우 -1과 1 사이)
- MinMaxScaler 이용
- 데이터 분포가 가우시안 분포가 아닐 경우 적용

```
from sklearn.preprocessing import MinMaxScaler

# MinMaxScaler객체 생성
scaler = MinMaxScaler()
# MinMaxScaler 로 데이터 셋 변환. fit() 과 transform() 호출.
scaler.fit(iris_df)
iris_scaled = scaler.transform(iris_df)
```

2.6 프로세스 정리

📌 데이터 전처리

Null 값 처리, scaling => sklearn.preprocessing 모듈

📌 train, test data 분리

=> sklearn.model_selection 모듈

📌 모델 학습 (train data 이용)

데이터 특성별 모델 선택

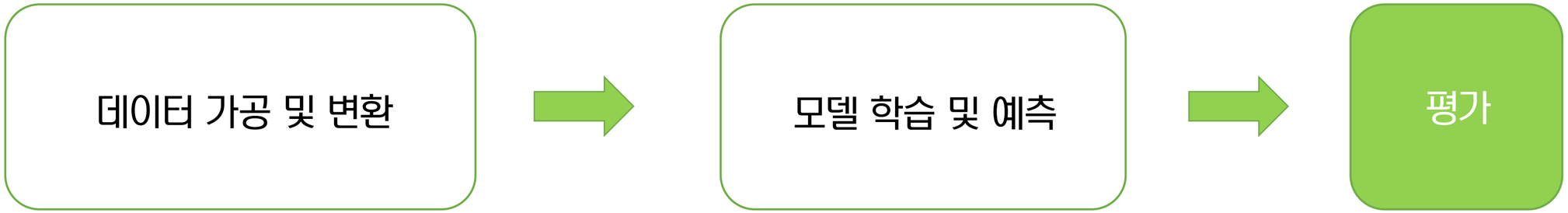
교차 검증 => sklearn.model_selection 모듈

📌 모델 성능 평가 (test data 이용)

03. 평가

EWHA EURON

분류 문제의 성능 평가 지표



회귀

: RMSE, MSE, MAE … ⇨ 실제 값과 예측 값의 '차이', 즉 오차 평균 값에 기반한 평가지표를 사용

분류

: 분류도 실제 label 과 예측 label 이 얼마나 정확하고 오류가 적게 발생하는가에 focus 를 두지만 분류 문제는 이진 분류/멀티분류, class imbalance 문제 등 고려할 부분이 많기 때문에, 단순히 정답을 맞췄는지만 가지고 판단하면 잘못된 평가 결과를 도출할 수 있다 👉 여러 평가 지표가 존재

3.1 정확도

$$Accuracy(\text{정확도}) = \frac{TN+TP}{TN+FP+FN+TP}$$

→ 예측결과와 실제값이 동일한 건수 / 전체 데이터 수

📌 이진분류에서 정확도는 ML 모델의 성능을 왜곡할 수 있다.

```python
import numpy as np
from sklearn.base import BaseEstimator

class MyDummyClassifier(BaseEstimator):
    # fit( ) 메소드는 아무것도 학습하지 않음.
    def fit(self, X , y=None):
        pass

    # predict( ) 메소드는 단순히 Sex feature가 1 이면 0 , 그렇지 않으면 1 로 예측함.
    def predict(self, X):
        pred = np.zeros( ( X.shape[0], 1 ))
        for i in range (X.shape[0]) :
            if X['Sex'].iloc[i] == 1:
                pred[i] = 0
            else :
                pred[i] = 1

        return pred
```

타이타닉 예제

Sex	Survived	
female	0	81
	1	233
male	0	468
	1	109

🗣 성별이 여성이면 1(생존), 남성이면 0(사망) 으로 예측해줘

```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# 원본 데이터를 재로딩, 데이터 가공, 학습데이터/테스트 데이터 분할.
titanic_df = pd.read_csv('./titanic_train.csv')
y_titanic_df = titanic_df['Survived']
X_titanic_df= titanic_df.drop('Survived', axis=1)
X_titanic_df = transform_features(X_titanic_df)
X_train, X_test, y_train, y_test=train_test_split(X_titanic_df, y_titanic_df, #
                                                  test_size=0.2, random_state=0)

# 위에서 생성한 Dummy Classifier를 이용하여 학습/예측/평가 수행.
myclf = MyDummyClassifier()
myclf.fit(X_train ,y_train)

mypredictions = myclf.predict(X_test)
print('Dummy Classifier의 정확도는: {0:.4f}'.format(accuracy_score(y_test , mypredictions)))
```

➡ 정확도 평가지표

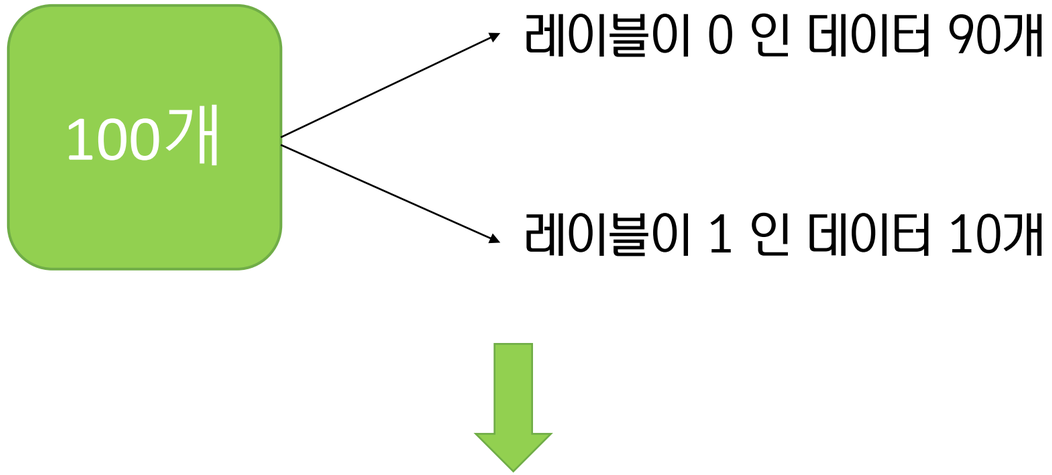
```
Dummy Classifier의 정확도는: 0.7877
```

너무 간단한 알고리즘에도 정확도 수치가 꽤 높게 나옴

3.1 정확도

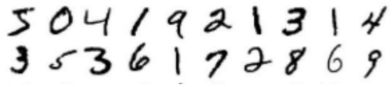
📌 이진분류에서 정확도는 ML 모델의 성능을 왜곡할 수 있다.

👀 **Imbalance 레이블 데이터**에서는 문제가 더 심각하다.



모두 0으로 맞춰도 정확도는
90% 가 됨

MNIST 예제



다중 레이블 문제

```
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split
from sklearn.base import BaseEstimator
from sklearn.metrics import accuracy_score
import numpy as np
import pandas as pd

class MyFakeClassifier(BaseEstimator):
    def fit(self,X,y):
        pass

    # 입력값으로 들어오는 X 데이터 셋의 크기만큼 모두 0값으로 만들어서 반환
    def predict(self,X):
        return np.zeros( (len(X), 1) , dtype=bool)

# 사이킷런의 내장 데이터 셋인 load_digits( )를 이용하여 MNIST 데이터 로딩
digits = load_digits()
```

🗣️ 전부다 0(False)으로 예측해줘
⇨ Target 예측값을 모두 False 레이블로 반환함[0,0,0,...0,0]

```
# digits번호가 7번이면 True이고 이를 astype(int)로 1로 변환, 7번이 아니면 False이고 0으로 변환.
y = (digits.target == 7).astype(int)
X_train, X_test, y_train, y_test = train_test_split( digits.data, y, random_state=11)
```

```
# 불균형한 레이블 데이터 분포도 확인.
print('레이블 테스트 세트 크기 :', y_test.shape)
print('테스트 세트 레이블 0 과 1의 분포도')
print(pd.Series(y_test).value_counts())

# Dummy Classifier로 학습/예측/정확도 평가
fakeclf = MyFakeClassifier()
fakeclf.fit(X_train , y_train)
fakepred = fakeclf.predict(X_test)
print('모든 예측을 0으로 하여도 정확도는:{:.3f}'.format(accuracy_score(y_test , fakepred)))
```

```
레이블 테스트 세트 크기 : (450,)
테스트 세트 레이블 0 과 1의 분포도
0    405
1     45
dtype: int64
모든 예측을 0으로 하여도 정확도는:0.900
```

accuracy
405 / 450 = 9/10

3.2 오차행렬

▧ 앞선 한계점을 극복하기 위해 나온 지표 ⇨ Confusion matrix

👀 학습을 잘한 것도 보여주고 못한 것도 보여주자!

example ☞ 사기 행위 예측 Positive : 사기 행위 Negetive : 정상 행위 ☞ 암 검진 예측 Positive : 양성 Negetive : 음성		실제 정답	
		True (Positive)	False (Negative)
분류 결과	True (Positive)	True Positive	False Positive Pos로 예측했는데 틀린거
	False (Negative)	False Negative Neg로 예측했는데 틀린거	True Negative

- $TN(TrueNegative)$: 실제 값이 Negative인데 예측 값도 Negative
- $FP(FalsePositive)$: 실제 값이 Negative인데 예측 값을 Positive
- $FN(FalseNegative)$: 실제 값이 Positive인데 예측 값을 Negative
- $TP(TruePositive)$: 실제 값이 Positive인데 예측 값도 Positive

MNIST 예제

```
from sklearn.metrics import confusion_matrix

# 앞절의 예측 결과인 fakepred와 실제 결과인 y_test의 Confusion Matrix출력
confusion_matrix(y_test , fakepred)
```

```
array([[405,   0],
       [ 45,   0]], dtype=int64)
```

```
True Negative :  405  --> (7이 아닌데 7이 아니라고 예측)
False Positive :  0 --> (7이 아닌데 7이라고 예측)
False Negative :  45 --> (7인데 7이 아니라고 예측)
True Positigve:  0 --> (7인데 7이라고 예측)
```

- $Accuracy$(정확도) $= \frac{TN+TP}{TN+FP+FN+TP}$

  → 예측결과와 실제값이 동일한 건수 / 전체 데이터 수

- $Precision$(정밀도) $= \frac{TP}{FP+TP}$

  → 예측대상(Positive)을 정확히 예측한 수 / Positive로 예측한 데이터 수

- $Recall$(재현율), $Sensitivity$(민감도), $TruePositiveRate(TPR)$ $= \frac{TP}{FN+TP}$

  → Positive를 정확히 예측한 수 / 전체 Positive 데이터 수

- $Specificity$(특이성), $TrueNegativeRate(TNR) = \frac{TN}{TN+FP}$

  → Negative를 정확히 예측한 수 / 전체 Negative 데이터 수

EWHA EURON

3.3 정밀도와 재현율

📌 accuracy 는 불균형한 이진 분류 문제에 취약하다 ⇨ 대안 : Precision, Recall

- $Precision(\text{정밀도}) = \frac{TP}{FP+TP}$

→ 예측대상(Positive)을 정확히 예측한 수 / Positive로 예측한 데이터 수

- $Recall(\text{재현율}), Sensitivity(\text{민감도}), TruePositiveRate(TPR)$
$= \frac{TP}{FN+TP}$

→ Positive를 정확히 예측한 수 / 전체 Positive 데이터 수

👀 Positive 데이터 세트의 예측 성능에 초점을 맞춘 지표

👉 실제 Negative 음성 데이터를 Positive 양성으로 잘못 판단하게 되면 큰 문제가 생기는 상황에 중요한 지표로 쓰임

(ex. 내가 교수님한테 보낸 메일이 스팸으로 분류되는 경우)

👉 FP (실제 Neg, 예측 Pos)를 낮추는데 초점

👉 실제 Positive 양성 데이터를 Negative 로 잘못 판단하게 되면 큰 문제가 생기는 상황에 중요한 지표로 쓰임

(ex. 암 진단 모델, 금융 사기 적발 모델

→ 사기거래를 정상 거래로 판단하는 경우)

👉 FN (실제 Pos, 예측 Neg)를 낮추는데 초점

👀 가장 좋은 성능 평가는 두 값이 모두 높은 수치인 경우!
BUT…

3.3 정밀도와 재현율

타이타닉 예제

오차행렬, 정확도, 정밀도, 재현율을 한꺼번에 계산하는 함수 생성

```
from sklearn.metrics import accuracy_score, precision_score , recall_score , confusion_matrix

def get_clf_eval(y_test , pred):
    confusion = confusion_matrix( y_test, pred)
    accuracy = accuracy_score(y_test , pred)
    precision = precision_score(y_test , pred)
    recall = recall_score(y_test , pred)
    print('오차 행렬')
    print(confusion)
    print('정확도: {0:.4f}, 정밀도: {1:.4f}, 재현율: {2:.4f}'.format(accuracy , precision ,recall))
```

로지스틱 회귀로 분류를 수행

```
import numpy as np
import pandas as pd

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression

# 원본 데이터를 재로딩, 데이터 가공, 학습데이터/테스트 데이터 분할.
titanic_df = pd.read_csv('./titanic_train.csv')
y_titanic_df = titanic_df['Survived']
X_titanic_df= titanic_df.drop('Survived', axis=1)
X_titanic_df = transform_features(X_titanic_df)

X_train, X_test, y_train, y_test = train_test_split(X_titanic_df, y_titanic_df, #
                                                    test_size=0.20, random_state=11)

lr_clf = LogisticRegression()

lr_clf.fit(X_train , y_train)
pred = lr_clf.predict(X_test)
get_clf_eval(y_test , pred)
```

로지스틱 회귀

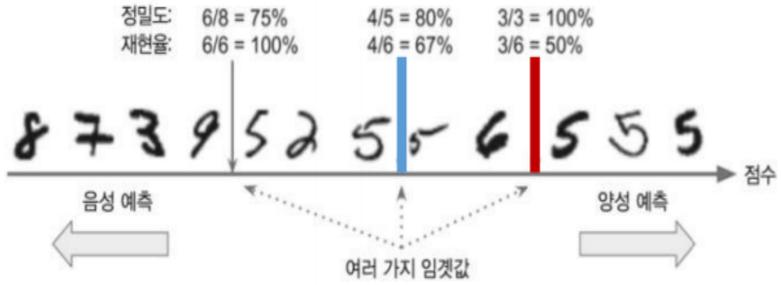
```
오차행렬
[[108  10]
 [ 14  47]]
정확도 : 0.8659
정밀도 : 0.8246
재현율 : 0.7705
```

재현율이 정밀도에 비해 낮게 나옴

📌 둘 다 성능을 올릴 수 있는 방법 없나? ☒

👉 정밀도/재현율 트레이드오프

정밀도/재현율 트레이드오프 : 정밀도와 재현율은 서로 상호 보완적인 평가 지표이기 때문에, 어느 하나를 높이면 다른 하나의 수치는 떨어지게 되는 것.



임계값: 정밀도 80% 재현율 67%

임계값: 정밀도 100% 재현율 50%

➔임계값을 올리면 정밀도 높아지고 재현율 낮아짐

반대로 임계값을 내리면 재현율 높아지고 정밀도 낮아짐

3.3 정밀도와 재현율

임계값

이진 분류 example

0 이 될 확률 (10%), 1 이 될 확률 (90%)

∴ label : 1

⇨ 일반적으로 임계값을 0.5로 설정하여

기준 값보다 확률이 크면 Pos, 작으면

Neg 로 결정한다.

임계값을 변경하면서
재현율과 정밀도 값의 변동을 살펴보자!

```
pred_proba = lr_clf.predict_proba(X_test)
pred  = lr_clf.predict(X_test)
print('pred_proba()결과 Shape : {0}'.format(pred_proba.shape))
print('pred_proba array에서 앞 3개만 샘플로 추출 \n:', pred_proba[:3])

# 예측 확률 array 와 예측 결과값 array 를 concatenate 하여 예측 확률과 결과값을 한눈에 확인
pred_proba_result = np.concatenate([pred_proba , pred.reshape(-1,1)],axis=1)
print('두개의 class 중에서 더 큰 확률을 클래스 값으로 예측 \n',pred_proba_result[:3])
```

로지스틱 회귀

```
pred_proba()결과 Shape : (179, 2)
pred_proba array에서 앞 3개만 샘플로 추출
: [[0.46162417 0.53837583]
 [0.87858538 0.12141462]
 [0.87723741 0.12276259]]
두개의 class 중에서 더 큰 확률을 클래스 값으로 예측
 [[0.46162417 0.53837583 1.        ]
 [0.87858538 0.12141462 0.        ]
 [0.87723741 0.12276259 0.        ]]
```

사이킷런은 predict_proba() 로 예측 확률을 반환하는 메서드를 제공한다.
학습이 완료된 사이킷런 분류 모델에서 불러올 수 있다.

분류 결정 임계값 **0.5** 기반에서 **Binarizer**를 이용하여 예측값 변환

```
오차행렬
[[108  10]
 [ 14  47]]
정확도 : 0.8659
정밀도 : 0.8246
재현율 : 0.7705
```

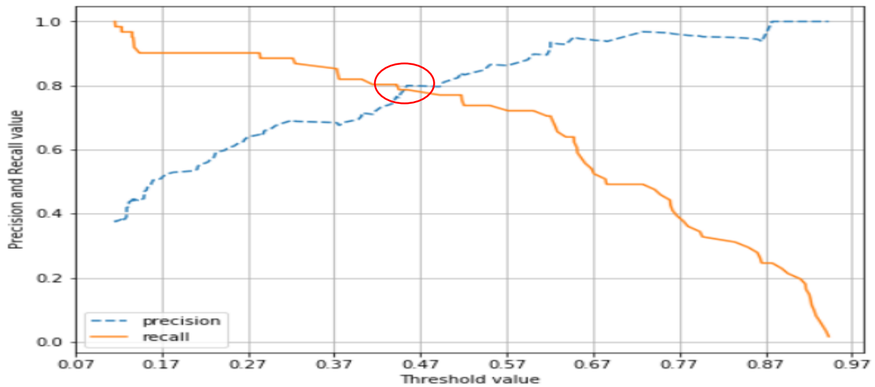
```
오차행렬
[[97 21]
 [11 50]]
정확도 : 0.8212
정밀도 : 0.7042
재현율 : 0.8197
```

분류 결정 임계값 **0.4** 기반에서 **Binarizer**를 이용하여 예측값 변환

→ threshold를 낮추니 정밀도는 떨어지고, 재현율이 올라감 (즉, 0.4부터 Positive로 예측을 하니, 전체 Positive 수 대비 Positive로 예측된 값의 수가 많아짐)

$$Precision(\text{정밀도}) = \frac{TP}{FP+TP} \quad \cdot \quad Recall(\text{재현율}) = \frac{TP}{FN+TP}$$

정밀도는 분모(Pos 로 예측한 데이터 수) 가 증가하고, 재현율은 (전체 Pos 데이터 수)로 분모값이 고정되는 격이기 때문에 분자 값이 똑같이 증가할 때, 정밀도는 감소하는 것이다.



양성 예측 수 증가 방향

3.4 F1 Score

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

▧ F1-score

: 정밀도와 재현율을 결합한 지표

정밀도와 재현율이 어느 한쪽으로 치우치지 않는 수치를 나타낼 때 상대적으로 높은 값을 가진다.

예시) 모델 A : 정밀도 0.9, 재현율이 0.1 , 모델 B : 정밀도 0.5, 재현율 0.5 일 때
→ 모델 A의 F1 = 0.18 , 모델 B의 F1 = 0.5

```python
from sklearn.metrics import f1_score
f1 = f1_score(y_test , pred)
print('F1 스코어: {0:.4f}'.format(f1))
```

F1 스코어: 0.7805

```python
def get_clf_eval(y_test , pred):
    confusion = confusion_matrix( y_test, pred)
    accuracy = accuracy_score(y_test , pred)
    precision = precision_score(y_test , pred)
    recall = recall_score(y_test , pred)
    # F1 스코어 추가
    f1 = f1_score(y_test,pred)
    print('오차 행렬')
    print(confusion)
    # f1 score print 추가
    print('정확도: {0:.4f}, 정밀도: {1:.4f}, 재현율: {2:.4f}, F1:{3:.4f}'.format(accuracy, precision, recall, f1))

thresholds = [0.4 , 0.45 , 0.50 , 0.55 , 0.60]
pred_proba = lr_clf.predict_proba(X_test)
get_eval_by_threshold(y_test, pred_proba[:,1].reshape(-1,1), thresholds)
```

```
임곗값: 0.4
오차 행렬
[[99 19]
 [10 51]]
정확도: 0.8380, 정밀도: 0.7286, 재현율: 0.8361, F1:0.7786
임곗값: 0.45
오차 행렬
[[103  15]
 [ 12  49]]
정확도: 0.8492, 정밀도: 0.7656, 재현율: 0.8033, F1:0.7840
임곗값: 0.5
오차 행렬
[[104  14]
 [ 13  48]]
정확도: 0.8492, 정밀도: 0.7742, 재현율: 0.7869, F1:0.7805
임곗값: 0.55
오차 행렬
[[109   9]
 [ 15  46]]
정확도: 0.8659, 정밀도: 0.8364, 재현율: 0.7541, F1:0.7931
임곗값: 0.6
오차 행렬
[[112   6]
 [ 16  45]]
정확도: 0.8771, 정밀도: 0.8824, 재현율: 0.7377, F1:0.8036
```

임계값을 변화하면서 살펴봤을 때 가장 F1 점수가 높은 경우! 그러나 재현율이 크게 감소하였으니 주의해야함

3.5 ROC 곡선과 AUC

ROC 곡선(수신기 조작 특성): 민감도(재현율)에 대한 1-특이도 그래프

AUC(area under the curve): ROC 곡선의 밑넓이

👀 2진분류에서 중요한 지표로 활용됨

민감도(재현율)
(true positive rate; TPR)
= TP/(FN+TP)

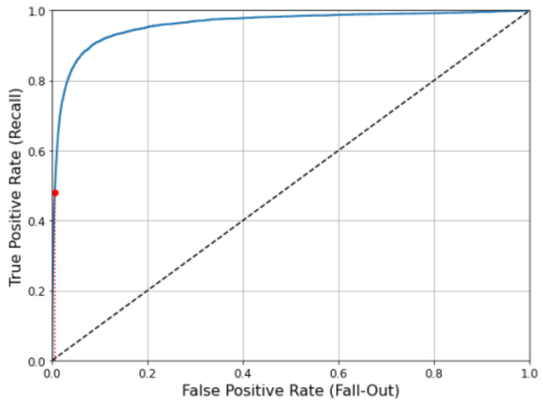
Pos 가 정확히 예측돼야 하는 수준
질병이 있는 사람을 질병 있다고 판단

특이도
(true negative rate; TNR)
= TN/TN+FP

Neg가 정확히 예측돼야 하는 수준
질병이 없는 사람을 질병 없다고 판단

roc_curve() 함수를 이용해 여러 임계값에서 fpr과 tpr계산

ROC 곡선

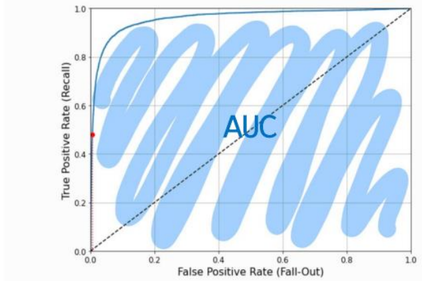



```
from sklearn.metrics import roc_curve

# 레이블 값이 1일때의 예측 확률을 추출
pred_proba_class1 = lr_clf.predict_proba(X_test)[:, 1]

fprs , tprs , thresholds = roc_curve(y_test,pred_proba_class)
```

```
def roc_curve_plot(y_test , pred_proba_c1):
    # 임곗값에 따른 FPR, TPR 값을 반환 받음.
    fprs , tprs , thresholds = roc_curve(y_test ,pred_proba_c1)

    # ROC Curve를 plot 곡선으로 그림.
    plt.plot(fprs , tprs, label='ROC')
    # 가운데 대각선 직선을 그림.
    plt.plot([0, 1], [0, 1], 'k--', label='Random')

    # FPR X 축의 Scale을 0.1 단위로 변경, X,Y 축명 설정등
    start, end = plt.xlim()
    plt.xticks(np.round(np.arange(start, end, 0.1),2))
    plt.xlim(0,1); plt.ylim(0,1)
    plt.xlabel('FPR( 1 - Sensitivity )'); plt.ylabel('TPR( Recall )')
    plt.legend()
    plt.show()

roc_curve_plot(y_test, lr_clf.predict_proba(X_test)[:, 1] )
```

roc_auc_score 함수를 이용해 AUC 계산

AUC(area under the curve): ROC 곡선의 밑넓이




```
from sklearn.metrics import roc_auc_score
```

```
pred_proba = lr_clf.predict_proba(X_test)[:, 1]
roc_score = roc_auc_score(y_test, pred_proba)
print('ROC AUC 값: {0:.4f}'.format(roc_score))
```

→ROC 곡선이 왼쪽 위 모서리와 가까울수록, AUC가 1에 가까울 수록 성능이 좋음

3.6 피마 인디언 당뇨병 예측

데이터 로딩

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline

from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score, roc_auc_score
from sklearn.metrics import f1_score, confusion_matrix, precision_recall_curve, roc_curve
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression

diabetes_data = pd.read_csv('diabetes.csv')
print(diabetes_data['Outcome'].value_counts())
diabetes_data.head(3)
```

```
0    500
1    268
Name: Outcome, dtype: int64
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1

```
diabetes_data.info( )
```

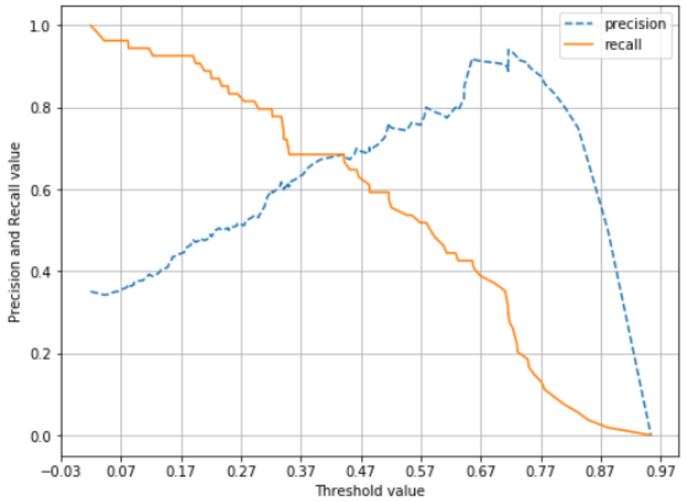
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
Pregnancies                 768 non-null int64
Glucose                     768 non-null int64
BloodPressure               768 non-null int64
SkinThickness               768 non-null int64
Insulin                     768 non-null int64
BMI                         768 non-null float64
DiabetesPedigreeFunction    768 non-null float64
Age                         768 non-null int64
Outcome                     768 non-null int64
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

모델 훈련 및 평가

```
# 피처 데이터 세트 X, 레이블 데이터 세트 y를 추출.
# 맨 끝이 Outcome 컬럼으로 레이블 값임. 컬럼 위치 -1을 이용해 추출
X = diabetes_data.iloc[:, :-1]
y = diabetes_data.iloc[:, -1]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 156, stratify=y)

# 로지스틱 회귀로 학습,예측 및 평가 수행.
lr_clf = LogisticRegression()
lr_clf.fit(X_train , y_train)
pred = lr_clf.predict(X_test)
pred_proba = lr_clf.predict_proba(X_test)[:, 1]

get_clf_eval(y_test , pred, pred_proba)
```

```
오차 행렬
[[87 13]
 [22 32]]
정확도: 0.7727, 정밀도: 0.7111, 재현율: 0.5926,    F1: 0.6465, AUC:0.8083
```

```
pred_proba_c1 = lr_clf.predict_proba(X_test)[:, 1]
precision_recall_curve_plot(y_test, pred_proba_c1)
```



3.6 피마 인디언 당뇨병 예측

널값(0) 대체

```
# 0값을 검사할 피처명 리스트 객체 설정
zero_features = ['Glucose', 'BloodPressure','SkinThickness','Insulin','BMI']
```

```
# zero_features 리스트 내부에 저장된 개별 피처들에 대해서 0값을 평균 값으로 대체
diabetes_data[zero_features]=diabetes_data[zero_features].replace(0, diabetes_data[zero_features].mean())
```

피처 스케일링 및 모델 훈련

```
X = diabetes_data.iloc[:, :-1]
y = diabetes_data.iloc[:, -1]

# StandardScaler 클래스를 이용해 피처 데이터 세트에 일괄적으로 스케일링 적용
scaler = StandardScaler( )
X_scaled = scaler.fit_transform(X)

X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size = 0.2, random_state = 156, stratify=y)

# 로지스틱 회귀로 학습, 예측 및 평가 수행.
lr_clf = LogisticRegression()
lr_clf.fit(X_train , y_train)
pred = lr_clf.predict(X_test)
pred_proba = lr_clf.predict_proba(X_test)[:, 1]

get_clf_eval(y_test , pred, pred_proba)
```

```
오차 행렬
[[90 10]
 [21 33]]
정확도: 0.7987, 정밀도: 0.7674, 재현율: 0.6111,    F1: 0.6804, AUC:0.8433
```

성능이 어느정도 개선됨

모델 평가

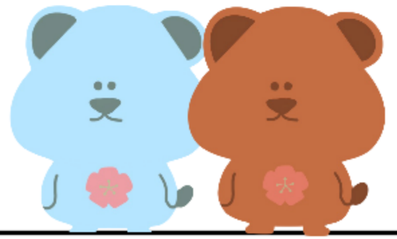
```
from sklearn.preprocessing import Binarizer

def get_eval_by_threshold(y_test , pred_proba_c1, thresholds):
    # thresholds 리스트 객체내의 값을 차례로 iteration하면서 Evaluation 수행.
    for custom_threshold in thresholds:
        binarizer = Binarizer(threshold=custom_threshold).fit(pred_proba_c1)
        custom_predict = binarizer.transform(pred_proba_c1)
        print('임곗값:',custom_threshold)
        get_clf_eval(y_test , custom_predict, pred_proba_c1)
```

```
thresholds = [0.3 , 0.33 ,0.36,0.39, 0.42 , 0.45 ,0.48, 0.50]
pred_proba = lr_clf.predict_proba(X_test)
get_eval_by_threshold(y_test, pred_proba[:,1].reshape(-1,1), thresholds )
```

```
임곗값: 0.48
오차 행렬
[[88 12]
 [19 35]]
정확도: 0.7987, 정밀도: 0.7447, 재현율: 0.6481,    F1: 0.6931, AUC:0.8433
```

임계값을 조절했을때 AUC 값은 모두 0.8433으로 동일했으므로, F1-score 가 가장 높은 0.48을 최종 임계값으로 설정함

THANK YOU



EWHA EURON