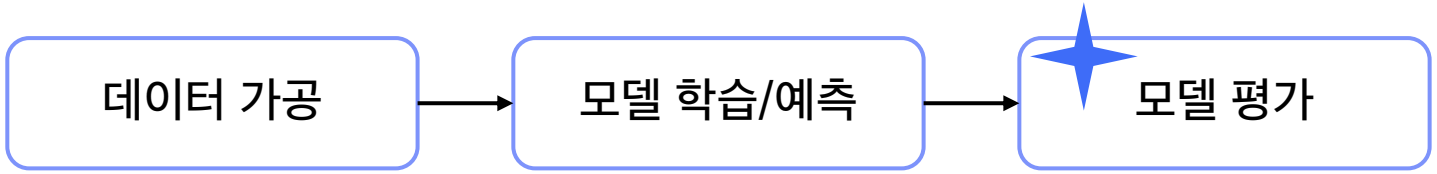
Aichemist Session

CHAP 03 평가

CONTENTS

평가

: 머신러닝 과정 중, 머신러닝 모델의 (**예측 성능**)을 평가하는 단계



→ 평가 방법? (**성능 평가 지표**)

분류	회귀
- 정확도(Accuracy) - 오차행렬(Confusion Matrix) - 정밀도(Precision) - 재현율(Recall) - F1 스코어 - ROC AUC → `(**이진분류**)의 성능 평가 지표` 중심	- MAE, MSE, RMSE 등 예측 오차 이용 → `CH 05 회귀` 에서 배울 예정!

이진 분류 (0,1,2개의 결정 클래스 값) vs 멀티(다중) 분류 (3개 이상의 결정 클래스 값)

01.

정확도

정확도

: 실제 데이터에서 예측 데이터가 얼마나 (같은) 지 판단하는 지표

$$\text{정확도 (Accuracy)} = \frac{\text{예측 결과가 동일한 데이터 건수}}{\text{전체 예측 데이터 건수}}$$

구현 함수 : accuracy_score(실제값, 예측값)

직관적.

그러나, (불균형한) 레이블 데이터 세트 가진 이진 분류 모델의 성능 왜곡 가능성 존재

ex) 교재 예제: 무조건 특정 값으로 예측해도 높은 정확도 나타날 수 있음

→ 다른 성능 평가 지표들도 같이 봐야 한다!

다음 교재 예제를 통해 맹점을 살펴보자.

정확도 실습1 - 타이타닉 생존자

(1) MyDummyClassifier 생성

(2) 데이터 전처리 함수 정의

(3) 학습/예측/평가

매우 단순한 분류 모델이어도 정확도가 높을 수 있다!

정확도 실습1 - 타이타닉 생존자

(1) MyDummyClassifier 생성

```
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
```

Sex = 1 ⇒ 0

Sex = 0 ⇒ 1

매우 단순한 분류 모델

(2) 데이터 전처리 함수 정의

```
# 데이터 전처리 함수 정의
from sklearn.preprocessing import LabelEncoder

# Null 처리 함수
def fillna(df):
    df['Age'].fillna(df['Age'].mean(), inplace=True)
    df['Cabin'].fillna('N', inplace=True)
    df['Embarked'].fillna('N', inplace=True)
    df['Fare'].fillna(0, inplace=True)
    return df

# 머신러닝 알고리즘에 불필요한 피처 제거
def drop_features(df):
    df.drop(['PassengerId', 'Name', 'Ticket'], axis=1, inplace=True)
    return df

# 레이블 인코딩 수행.
def format_features(df):
    df['Cabin'] = df['Cabin'].str[:1]
    features = ['Cabin', 'Sex', 'Embarked']
    for feature in features:
        le = LabelEncoder()
        le = le.fit(df[feature])
        df[feature] = le.transform(df[feature])
    return df

# 앞에서 설정한 데이터 전처리 함수 호출
def transform_features(df):
    df = fillna(df)
    df = drop_features(df)
    df = format_features(df)
    return df
```

정확도 실습1 - 타이타닉 생존자

(3) 학습/예측/평가

```
# 학습/예측/평가
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# 원본 데이터를 재로딩, 데이터 가공, 학습 데이터/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 이용해 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)))
```

```
Dummy Classifier의 정확도는: 0.7877
```

→ 매우 단순한 분류 모델임에도 정확도가 꽤 높음

정확도를 평가 지표로 사용할 때는 신중해야.

정확도 실습2 - MNIST

(1) MyFakeClassifier 정의, 데이터 준비

(2) 데이터 분포도 확인

(3) 학습/예측/평가

불균형한 레이블 데이터 세트에서 정확도의 맹점

정확도 실습2 - MNIST

(1) MyFakeClassifier 정의, 데이터 준비

```
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

# 다 0으로 예측하는 Classifier
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()

# 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)
```

- astype(): Numpy 라이브러리에서 배열의 데이터 타입 변경할 때 사용
- digits.target == 7: digits.target 배열에서 숫자 7이면 True, 아니면 False
- astype(int): True를 1로, False를 0으로 변환
- y: 7이면 타겟 레이블이 1인 이진 분류 레이블 배열

```
digits.target == 7
```

```
array([False, False, False, ..., False, False, False])
```

(2) 데이터 분포도 확인

```
# 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())
```

```
레이블 테스트 세트 크기 : (450,)
테스트 세트 레이블 0 과 1의 분포도
0    405
1     45
dtype: int64
```

불균형한 데이터 세트 생성 완료

정확도 실습2 - MNIST

(3) 학습/예측/평가

```
# 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)))
```

```
모든 예측을 0으로 하여도 정확도는:0.900
```

단순한 예측에도 매우 높은 정확도 보임

불균형한 레이블 데이터 세트에서 정확도를 성능 평가 수치로 사용하면 안된다.

02.

오차행렬

오차행렬

: 학습된 분류 모델의 예측 오류가 어떤 유형으로, 얼마나 발생하는지 함께 나타내는 지표

		예측값 Negative(0)	예측값 Positive(1)
실제값	Negative(0)	TN (True Negative)	FP (False Positive)
실제값	Positive(1)	FN (False Negative)	TP (True Positive)

구현 함수 : confusion_matrix(실제값, 예측값)

TN : Negative 로 예측, 예측이 맞음

FN : Negative 로 예측, 예측이 틀림

TP : Positive 로 예측, 예측이 맞음

FP : Positive 로 예측, 예측이 틀림

N/P (예측값)

T/F (예측이 맞음/틀림)

오차행렬 실습

```
# 오차행렬
from sklearn.metrics import confusion_matrix

# 정확도 실습2의 예측 결과인 fakepred와 실제 결과인 y_test의 Confusion Matrix출력
confusion_matrix(y_test , fakepred)
```

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

		예측값 Negative(0)	예측값 Positive(1)
실제값	Negative(0)	TN (405)	FP (0)
	Positive(1)	FN (45)	TP (0)

다 0으로 예측하여 FP, TP 값이 0임

오차행렬을 이용한 정확도(Accuracy) 재정의

		예측값 Negative	예측값 Positive
실제값	Negative	TN (True Negative)	FP (False Positive)
	Positive	FN (False Negative)	TP (True Positive)

$$\text{정확도 (Accuracy)} = \frac{\text{예측 결과가 동일한 데이터 건수}}{\text{전체 예측 데이터 건수}} = \frac{TN + TP}{TN + FP + FN + TP}$$

정확도의 맹점

(불균형한) 레이블 데이터 세트 가진 이진 분류 모델의 성능 왜곡 가능성

보통, 중점적으로 찾아야 하는 매우 적은 수의 결괏값에 Positive(1) 설정

→ 반대로, Negative가 많아짐.

Negative(0) 로 예측 정확도가 높아지는 경향 발생

→ Positive에 대한 예측 정확도 판단하지 못한 채

Negative에 대한 예측 정확도만으로도

분류의 정확도가 (높게) 나타날 수 있다.

→ 정밀도(Precision), 재현율(Recall)

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

03.

정밀도와 재현율

정밀도와 재현율

: (Positive) 클래스의 예측 성능에 더 초점을 맞춘 평가 지표, 불균형 데이터 세트에서 선호됨

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

- (예측)이 Positive인 대상 중 예측과 실제값이 Positive로 일치한 데이터 비율
- TP↑, FP↓에 초점
- 실제 Negative 인 데이터 예측을 Positive로 잘못 판단 시 큰 영향 발생하는 경우 사용
- precision_score(실제값, 예측값)

$$\text{재현율 (Recall)} = \frac{TP}{FN + TP}$$

- (실제값)이 Positive인 대상 중 예측과 실제값이 Positive로 일치한 데이터 비율
- TP↑, FN↓에 초점
- 실제 Positive 인 데이터 예측을 Negative로 잘못 판단 시 큰 영향 발생하는 경우 사용
- recall_score(실제값, 예측값)

정밀도/재현율 실습 - 타이타닉

(1) 평가 함수 get_clf_eval 정의

```
# 평가 함수 정의
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))
```

정밀도/재현율 실습 - 타이타닉

(2) 로지스틱 회귀 모델로 학습/예측/평가

```
# 타이타닉 데이터를 로지스틱 회귀 모델로 학습/예측/평가
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)
# 이전 타이타닉 예제에서의 전처리 함수 사용
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)

# 모델 생성
lr_clf = LogisticRegression(solver='liblinear')

# 학/예/평
lr_clf.fit(X_train , y_train)
pred = lr_clf.predict(X_test)
get_clf_eval(y_test , pred)
```

```
오차 행렬
[[108  10]
 [ 14  47]]
정확도: 0.8659, 정밀도: 0.8246, 재현율: 0.7705
```

- 데이터 로딩
- 이전 예제의 transform_features 함수로 데이터 전처리
- 학습/테스트용 데이터 분리
- 모델 생성
- 학습/예측/평가

정밀도/재현율 트레이드오프

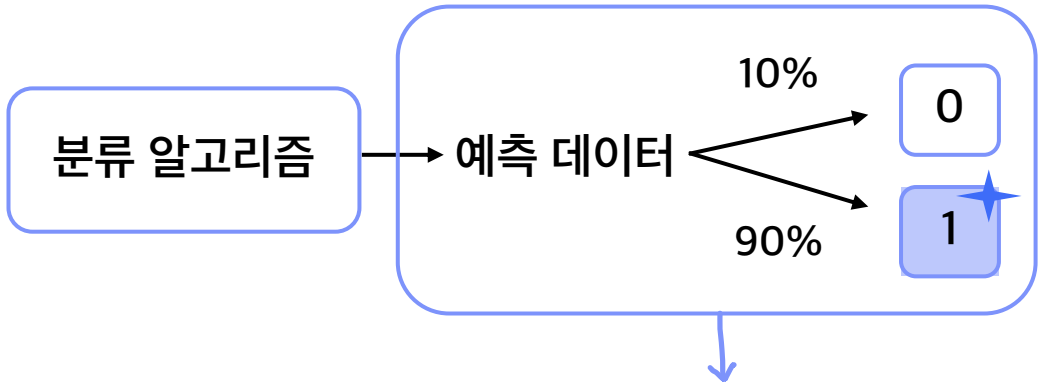
서로 (상호 보완적인) 평가 지표

둘 다 높은 수치를 얻는 게 좋은 평가

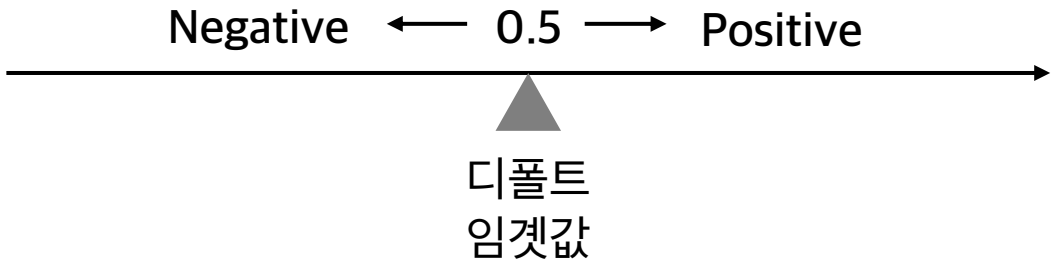
둘 중 하나를 특별히 강조해야 할 경우, (분류의 결정 임곗값) (Threshold) 조정해 수치 조절

→ 한 수치를 강제로 높이면 다른 수치가 떨어지기 쉬움

→ 정밀도/재현율 트레이드오프(Trade-off)



구현함수 : predict_proba(테스트 피처 데이터 세트)



임곗값: 1(Positive)일 확률 기준

트레이드오프 이해 (임곗값 조정에 따른 평가 지표 변화)

(1) predict_proba(테스트 피처 데이터 세트) : 개별 데이터별로 예측 확률 반환 메서드

```
# predict_proba 와 predict 비교
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(pred)와 예측 결과값 array(pred_proba)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])
```

앞 예제 노트북 이어서

```
pred_proba()결과 Shape : (179, 2)
pred_proba array에서 앞 3개만 샘플로 추출
: [[0.44935228 0.55064772]
 [0.86335513 0.13664487]
 [0.86429645 0.13570355]]
두개의 class 중에서 더 큰 확률을 클래스 값으로 예측
 [[0.44935228 0.55064772 1.        ]
 [0.86335513 0.13664487 0.        ]
 [0.86429645 0.13570355 0.        ]]
```

```
: [[0.44935228 0.55064772]
 [0.86335513 0.13664487]
 [0.86429645 0.13570355]]
```

```
2 1.
 0.
 0.
```

pred_proba: 첫 칼럼이 0일 예측 확률,

두번째 칼럼이 1일 예측 확률

predict: 더 큰 확률 값으로 최종 예측

트레이드오프 이해 (임곗값 조정에 따른 평가 지표 변화)

(2) Binarizer 클래스

- Binarizer(threshold) 객체 생성
- 객체.fit_transform(ndarray): 입력된 ndarray의 값을 threshold 값보다 작으면 0, 크면 1로 변환해 반환

```
# Binarizer 클래스 사용 예시
from sklearn.preprocessing import Binarizer

X = [[ 1, -1,  2],
     [ 2,  0,  0],
     [ 0,  1.1, 1.2]]

# threshold 기준값보다 같거나 작으면 0을, 크면 1을 반환
binarizer = Binarizer(threshold=1.1)
print(binarizer.fit_transform(X))
```

```
[[0. 0. 1.]
 [1. 0. 0.]
 [0. 0. 1.]]
```

트레이드오프 이해 (임곗값 조정에 따른 평가 지표 변화)

(3) predict() 의사 코드 (predict_proba와 Binarizer로 구현)

```
# predict 의사코드
from sklearn.preprocessing import Binarizer

#Binarizer의 threshold 설정값=분류 결정 임곗값
custom_threshold = 0.5

# predict_proba( ) 반환값의 두번째 컬럼 , 즉 Positive 클래스 컬럼 하나만 추출하여 Binarizer를 적용
pred_proba_1 = pred_proba[:,1].reshape(-1,1)

binarizer = Binarizer(threshold=custom_threshold).fit(pred_proba_1)
custom_predict = binarizer.transform(pred_proba_1)

get_clf_eval(y_test, custom_predict)
```

```
오차 행렬
[[108  10]
 [ 14  47]]
정확도: 0.8659, 정밀도: 0.8246, 재현율: 0.7705
```

앞 예제의 predict() 결과와 지표 정확히 일치

트레이드오프 이해 (임곗값 조정에 따른 평가 지표 변화)

(4) 분류 결정 임곗값 조정

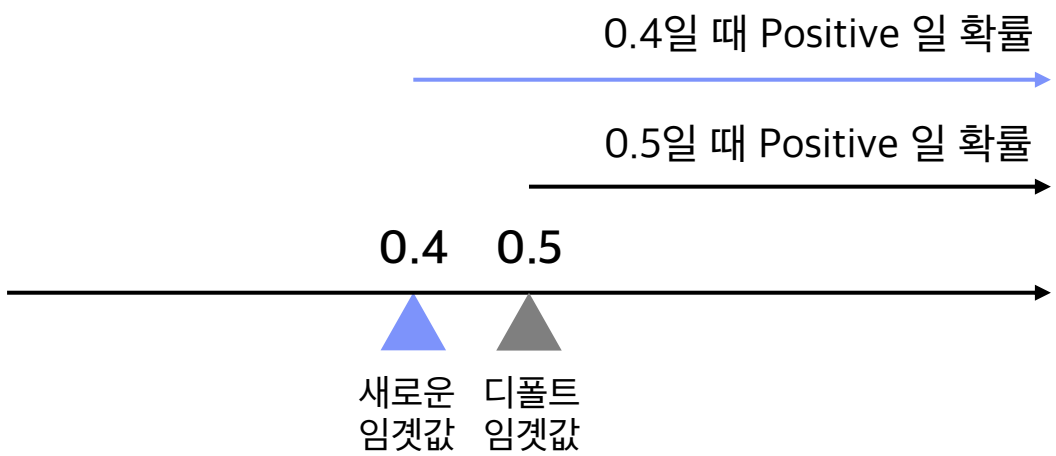
```
# 테스트를 수행할 모든 임곗값을 리스트 객체로 저장.
thresholds = [0.4, 0.45, 0.50, 0.55, 0.60]

def get_eval_by_threshold(y_test , pred_proba_c1, thresholds):
    # thresholds list객체내의 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)

get_eval_by_threshold(y_test ,pred_proba[:,1].reshape(-1,1), thresholds )
```

```
임곗값: 0.4
오차 행렬
[[97 21]
 [11 50]]
정확도: 0.8212, 정밀도: 0.7042, 재현율: 0.8197
임곗값: 0.45
오차 행렬
[[105  13]
 [ 13  48]]
정확도: 0.8547, 정밀도: 0.7869, 재현율: 0.7869
임곗값: 0.5
오차 행렬
[[108  10]
 [ 14  47]]
정확도: 0.8659, 정밀도: 0.8246, 재현율: 0.7705
임곗값: 0.55
오차 행렬
[[111   7]
 [ 16  45]]
정확도: 0.8715, 정밀도: 0.8654, 재현율: 0.7377
임곗값: 0.6
오차 행렬
[[113   5]
 [ 17  44]]
정확도: 0.8771, 정밀도: 0.8980, 재현율: 0.7213
```

분류 결정 임곗값 낮추니,

정밀도 (↓), 재현율 (↑)

임곗값(Threshold) 조정



임곗값: 1(Positive)일 확률 기준

임곗값 ↓

→ Positive로 예측 (↓/↑)

→ True 값 (↓/↑)

→ FN (↓/↑)

→ 정밀도↓, 재현율↑

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

$$\text{재현율 (Recall)} = \frac{\uparrow TP}{\downarrow FN + TP \uparrow}$$

트레이드오프 이해 (임곗값 조정에 따른 평가 지표 변화)

(5) precision_recall_curve()

앞에서 구현한 get_eval_by_threshold() 유사한 API

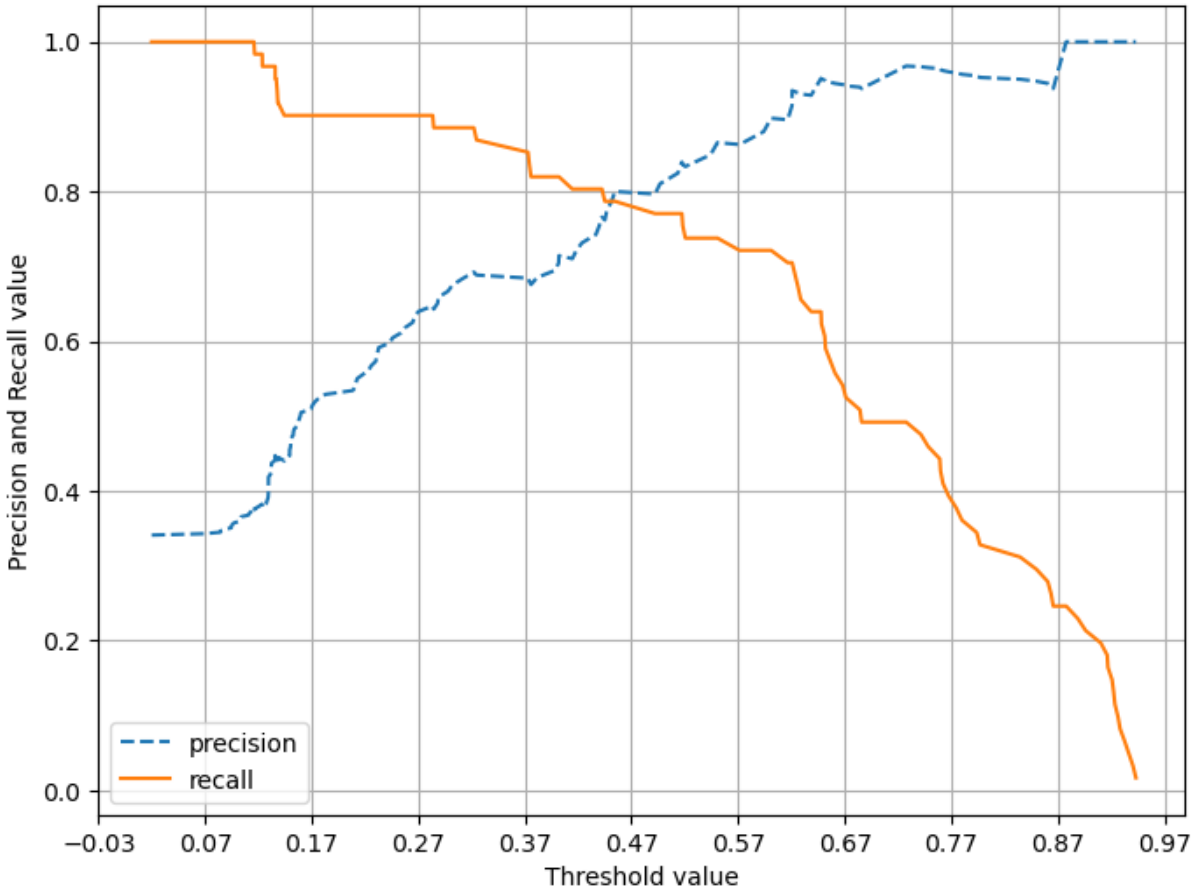
입력 파라미터	y_true: 실제 클래스값 배열 probas_pred: Positive 칼럼의 예측 확률 배열
반환값	정밀도, 재현율 : 값을 임곗값 별 각각 배열로 반환

```
# precision_recall_curve()
from sklearn.metrics import precision_recall_curve

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

# 실제값 데이터 셋과 레이블 값이 1일 때의 예측 확률을 precision_recall_curve 인자로 입력
precisions, recalls, thresholds = precision_recall_curve(y_test, pred_proba_class1 )
print('반환된 분류 결정 임곗값 배열의 Shape:', thresholds.shape)
print('반환된 precisions 배열의 Shape:', precisions.shape)
print('반환된 recalls 배열의 Shape:', recalls.shape)

print("thresholds 5 sample:", thresholds[:5])
print("precisions 5 sample:", precisions[:5])
print("recalls 5 sample:", recalls[:5])

#반환된 임계값 배열 로우가 147건이므로 샘플로 10건만 추출하되, 임곗값을 15 Step으로 추출.
thr_index = np.arange(0, thresholds.shape[0], 15)
print('샘플 추출을 위한 임계값 배열의 index 10개:', thr_index)
print('샘플용 10개의 임곗값: ', np.round(thresholds[thr_index], 2))

# 15 step 단위로 추출된 임계값에 따른 정밀도와 재현율 값
print('샘플 임계값별 정밀도: ', np.round(precisions[thr_index], 3))
print('샘플 임계값별 재현율: ', np.round(recalls[thr_index], 3))
```

```
반환된 분류 결정 임곗값 배열의 Shape: (165,)
반환된 precisions 배열의 Shape: (166,)
반환된 recalls 배열의 Shape: (166,)
thresholds 5 sample: [0.01974987 0.06956413 0.08402808 0.08474207 0.0892016 ]
precisions 5 sample: [0.34078212 0.34269663 0.34463277 0.34659091 0.34857143]
recalls 5 sample: [1. 1. 1. 1. 1.]
샘플 추출을 위한 임계값 배열의 index 10개: [  0  15  30  45  60  75  90 105 120 135 150]
샘플용 10개의 임곗값:  [0.02 0.11 0.13 0.14 0.16 0.24 0.32 0.45 0.62 0.73 0.87]
샘플 임계값별 정밀도:  [0.341 0.372 0.401 0.44  0.505 0.598 0.688 0.774 0.915 0.968 0.938]
샘플 임계값별 재현율:  [1.    1.    0.967 0.902 0.902 0.902 0.869 0.787 0.705 0.492 0.246]
```

트레이드오프 이해 (임곗값 조정에 따른 평가 지표 변화)

(6) 시각화



정밀도와 재현율의 맹점

- 임곗값에 따라 정밀도, 재현율 조정 가능
- 하나만 올리는 극단적 수치 조작이 가능하나,
  그건 성능이 나쁜 분류임!
- 두 수치를 (상호 보완)할 수 있는 수준에서 적용,
  단순히 하나를 높이기 위한 수단으로 사용X

04.

F1 스코어

F1 스코어

: (정밀도)와 (재현율)을 결합한 지표

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

정밀도, 재현율이 한쪽으로 (치우치지) 않는 수치를 나타낼 때 상대적으로 높은 값 가짐

구현 함수 : f1_score(실제값, 예측값)

F1 스코어 실습 - 타이타닉 (앞 예제 이어서)

(1) 사용법

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

```
F1 스코어: 0.7966
```

(2) 평가 함수에 F1 스코어 추가

```
# 평가 함수에 F1 스코어 추가
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
[[97 21]
 [11 50]]
정확도: 0.8212, 정밀도: 0.7042, 재현율: 0.8197, F1:0.7576
임곗값: 0.45
오차 행렬
[[105  13]
 [ 13  48]]
정확도: 0.8547, 정밀도: 0.7869, 재현율: 0.7869, F1:0.7869
임곗값: 0.5
오차 행렬
[[108  10]
 [ 14  47]]
정확도: 0.8659, 정밀도: 0.8246, 재현율: 0.7705, F1:0.7966
임곗값: 0.55
오차 행렬
[[111   7]
 [ 16  45]]
정확도: 0.8715, 정밀도: 0.8654, 재현율: 0.7377, F1:0.7965
임곗값: 0.6
오차 행렬
[[113   5]
 [ 17  44]]
정확도: 0.8771, 정밀도: 0.8980, 재현율: 0.7213, F1:0.8000
```

05.

ROC 곡선과 AUC

ROC 곡선과 AUC

ROC 곡선

: (FPR)이 변할 때 (TPR)이 어떻게 변하는지를 나타내는 곡선 구현 함수 : roc_curve(실제값, 예측확률)

- FPR(False Positive Rate)
- TPR(True Positive Rate) : 민감도, 실제값 Positive가 정확히 예측돼야 하는 수준
- TNR(True Negative Rate) : 특이성, 실제값 Negative가 정확히 예측돼야 하는 수준

$$TNR = \frac{TN}{FP + TN}$$

FPR = FP / (FP + TN) = 1 - TNR = 1 - 특이성

ROC 곡선과 AUC

FPR 0~1로 변경하며 TPR 변화 값을 구함

FPR = FP / (FP + TN)

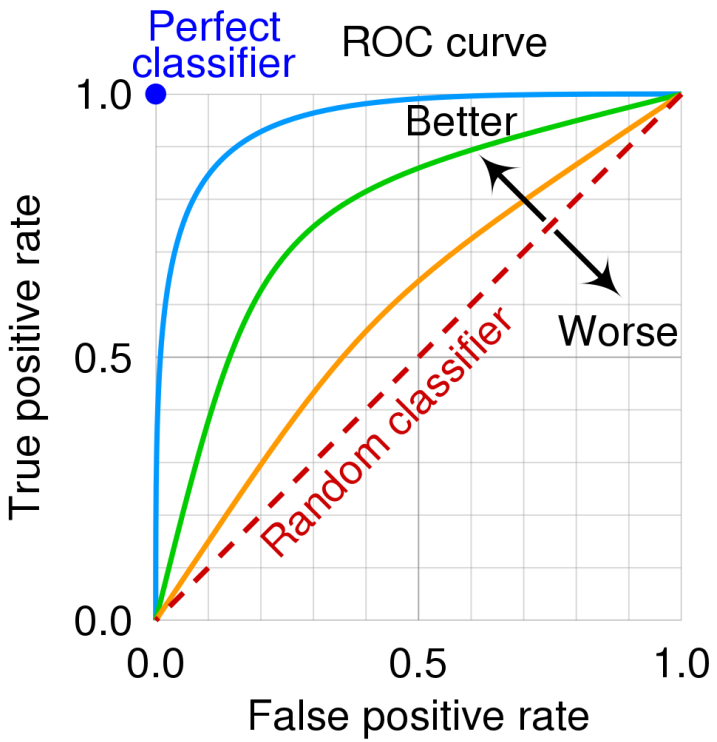
- FPR = 0 : 임곗값 1 지정 → FP = 0

- FPR = 1 : 임곗값 0 지정 → TN = 0

- **AUC**

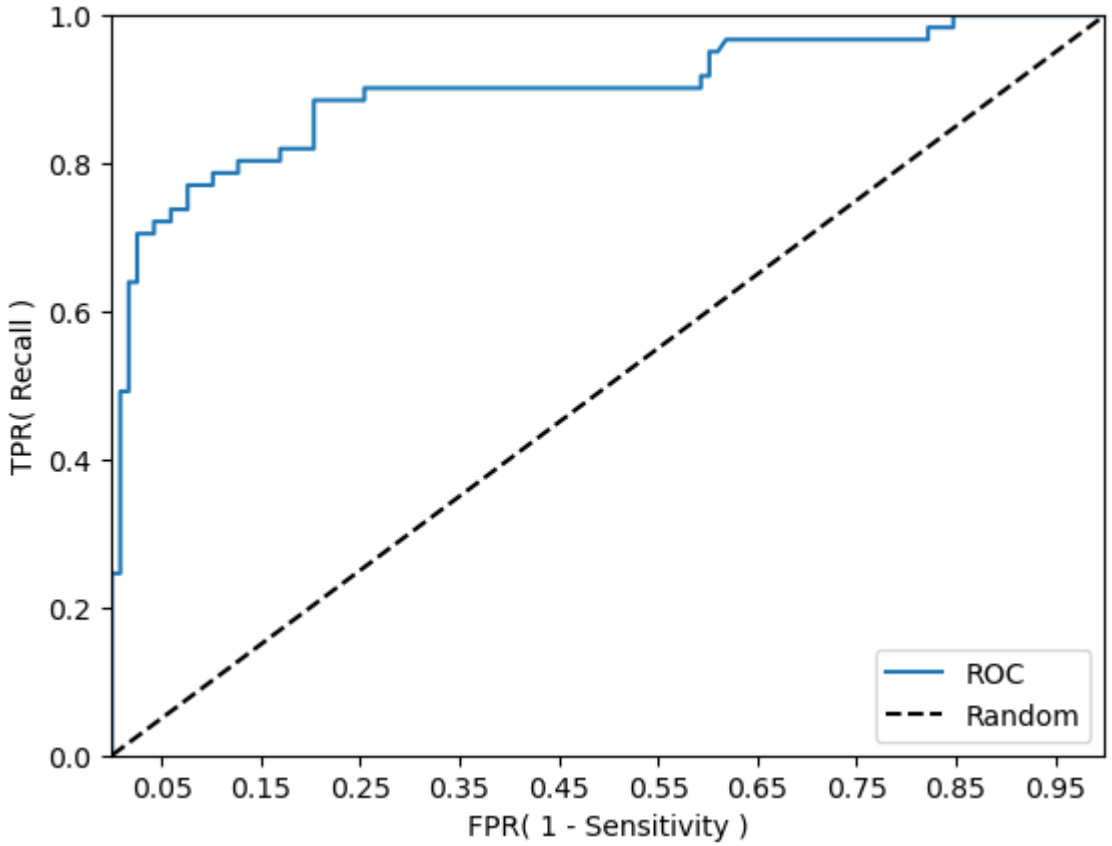
: ROC 곡선 밑의 (**면적**)

분류의 성능 지표, (**1**)에 가까울수록 좋은 수치



ROC-AUC 실습 - 타이타닉 (앞 예제 이어서)

(1) ROC 곡선 시각화



(2) 사용법

```
# roc_auc 사용
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 AUC 값: 0.8987
```

(3) 평가 함수 get_clf_eval 완성 (앞으로 사용!)

```
def get_clf_eval(y_test, pred=None, pred_proba=None):
    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 = f1_score(y_test,pred)
    # ROC-AUC 추가
    roc_auc = roc_auc_score(y_test, pred_proba)
    print('오차 행렬')
    print(confusion)
    # ROC-AUC print 추가
    print('정확도: {0:.4f}, 정밀도: {1:.4f}, 재현율: {2:.4f},₩
          F1: {3:.4f}, AUC:{4:.4f}'.format(accuracy, precision,
                                           recall, f1, roc_auc))
```

수고하셨습니다