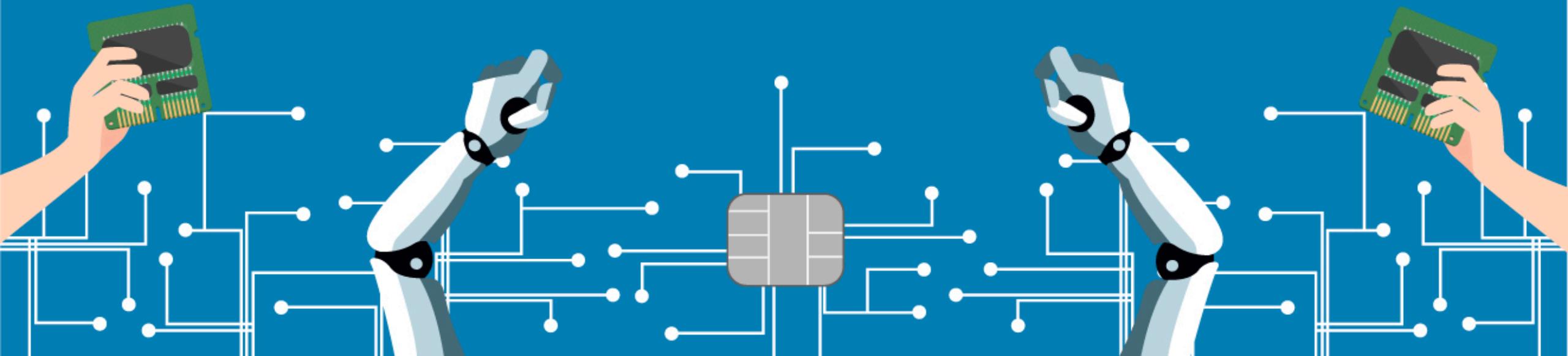


Kuggle

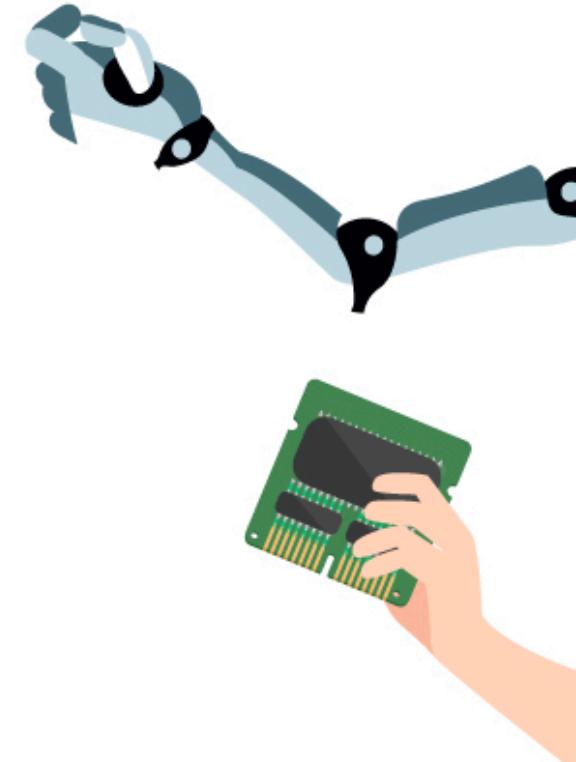
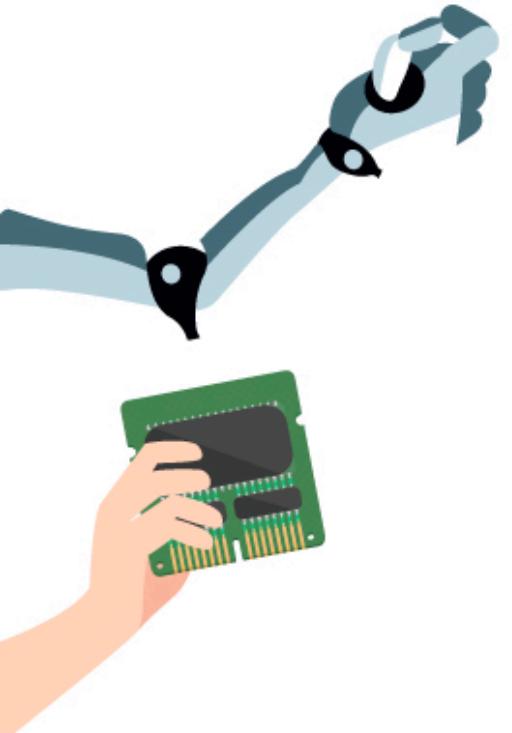
2021_12_28 Kuggle 방학정규세션_W1

2021.12.28



개요

- 1 머신러닝 : 전처리한 데이터 -> 모델 학습/예측 -> 평가
- 2 성능 평가 지표 : 회귀/분류
- 3 이진 분류



CONTENTS

01 오차 행렬

-오차행렬이란?

02 정확도

-Accuracy란?

-Accuracy Paradox

-불균형한 데이터셋

03 정밀도, 재현율

-정밀도, 재현율 정의

-정밀도, 재현율 코드

-트레이드오프

-Threshold 조정 코드

04 F1 Score

-F1 Score란?

-조화 평균

-F1 Score 코드

05 ROC, AUC

-TPR/FPR

-ROC 곡선 위 점

-ROC 곡선 훈정도

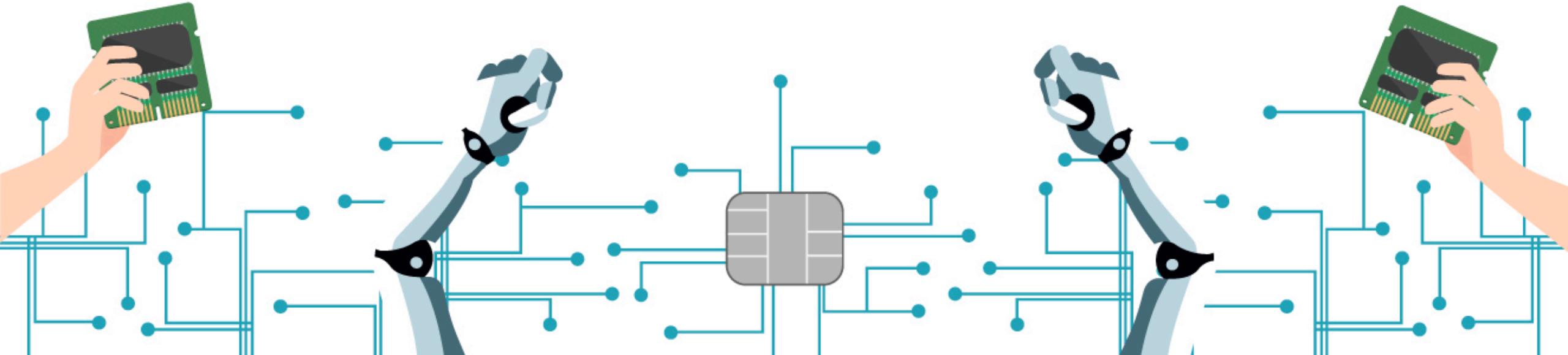
-코드 구현

06 데이터 실습

-어떤 성능 지표?

-전처리 후 성능지표

-Threshold 변경



CONTENTS

01. 오차 행렬

- 오차행렬이란?

01 오차 행렬

오차 행렬이란?

- 실제로 비가 온다 (True) vs 비가 오지 않는다 (False)
- 우산을 챙긴다 (Positive) vs 챙기지 않는다 (Negative)

실제

예측

	Negative (우산을 안 챙긴다)	Positive (우산을 챙긴다)
False (비가 안온다)	True Negative	False Positive
True (비가 온다)	False Negative	True Positive

CONTENTS

02. Accuracy

- Accuracy란
- Accuracy Paradox
- 불균형한 데이터 셋 -> code

02 정확도

Accuracy란?

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

=

$$\frac{\text{TruePositives} + \text{TrueNegatives}}{\text{TruePositives} + \text{TrueNegatives} + \text{FalsePositives} + \text{FalseNegatives}}$$

02 정확도

Accuracy란?

- 실제로 비가 온다 (True) vs 비가 오지 않는다 (False)
- 우산을 챙긴다 (Positive) vs 챙기지 않는다 (Negative)

	Negative (우산을 안 챙긴다)	Positive (우산을 챙긴다)
False (비가 안온다)	True Negative	False Positive
True (비가 온다)	False Negative	True Positive

02 정확도

Accuracy Paradox

	Negative (사기 안당할 것이라 예측)	Positive (사기 당할 것이라 예측)
False (사기 안당함)	990	0
True (사기 당함)	10	0

02 정확도

남자->사망, 여자->생존으로 알고리즘을 설정해도 Accuracy=0.78

```
import numpy as np
from sklearn.base import BaseEstimator #LinearRegression() 처럼 클래스를 임의로 설정,

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 1로 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
```

	Pclass	Sex	Age	SibSp	Parch	Fare	Cabin	Embarked
495	3	1	29.699118	0	0	14.4583	7	0
648	3	1	29.699118	0	0	7.5500	7	3
278	3	1	7.000000	4	1	29.1250	7	2
31	1	0	29.699118	1	0	146.5208	1	0
255	3	0	29.000000	0	2	15.2458	7	0

```
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: {:.4f}'.format(accuracy_score(y_test, mypredictions)))

Dummy Classifier의 정확도는: 0.7877
```

```
array([[0.],
       [0.],
       [0.],
       [1.],
       [1.],
```

02 정확도

불균형 데이터셋을 만든 후 Accuracy 확인

```
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 = 70이 아니다
    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)
```

```
# 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
```

CONTENTS

03. Recall, Precision

- 정밀도, 재현율 정의
- 정밀도, 재현율 코드구현
- 정밀도/재현율 트레이드오프
- Threshold 조정 코드

03 정밀도, 재현율

-재현율

재현율이란?

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

	Negative (사기 안당할 것이라 예측)	Positive (사기 당할 것이라 예측)
False (사기 안당함)	990	0
True (사기 당함)	10	TP

03 정밀도, 재현율

-재현율

재현율이 중요 지표일 경우?

	Negative (암환자 아니라 예측)	Positive (암환자라 예측)
False (암환자 아닌 경우)	980	10
True (암환자인 경우)	8	2
	FN	TP

03 정밀도, 재현율

-정밀도

정밀도란?

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

	Negative (사기 안당할 것이라 예측)	Positive (사기 당할 것이라 예측)
False (사기 안당함)	0	990 FP
True (사기 당함)	0	10 TP

03 정밀도, 재현율

-정밀도

정밀도가 중요 지표일 경우?

	Negative (스팸메일이 아니라 예측)	Positive (스팸메일이라 예측)
False (스팸 아닌 경우)	980	10 FP
True (스팸메일인 경우)	8	2 TP

03 정밀도, 재현율

```
#get_clf_eval 함수 설정
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) #y_test와 예측값
    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: {:.4f}, 정밀도: {:.4f}, 재현율: {:.4f}'.format(accuracy, precision, recall))
```

```
# 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() #logistic

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

```
오차 행렬
[[104 14]
 [ 13 48]]
정확도: 0.8492, 정밀도: 0.7742, 재현율: 0.7869
```

03 정밀도, 재현율

Predict로 인해 예측되는 0,1의 값은 어떻게?

```
lr_clf = LogisticRegression() #logistic
lr_clf.fit(X_train, y_train)
pred = lr_clf.predict(X_test)
get_clf_eval(y_test, pred) #get_clf_eval에 적용
```

```
pred_proba = lr_clf.predict_proba(X_test) #0or1에 속할 확률
pred = lr_clf.predict(X_test) #확률의 크기를 가지고 선택
print('pred_proba()결과 Shape : {}'.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])
```

```
pred_proba()결과 Shape : (179, 2)
pred_proba array에서 앞 3개만 샘플로 추출
: [[0.46196457 0.53803543]
 [0.87861802 0.12138198]
 [0.8771453 0.12228547 ]]
두개의 class 중에서 더 큰 확률을 클래스 값으로 예측
[[0.46196457 0.53803543 1.          ]
 [0.87861802 0.12138198 0.          ]
 [0.8771453 0.12228547 0.          ]]
```

03 정밀도, 재현율

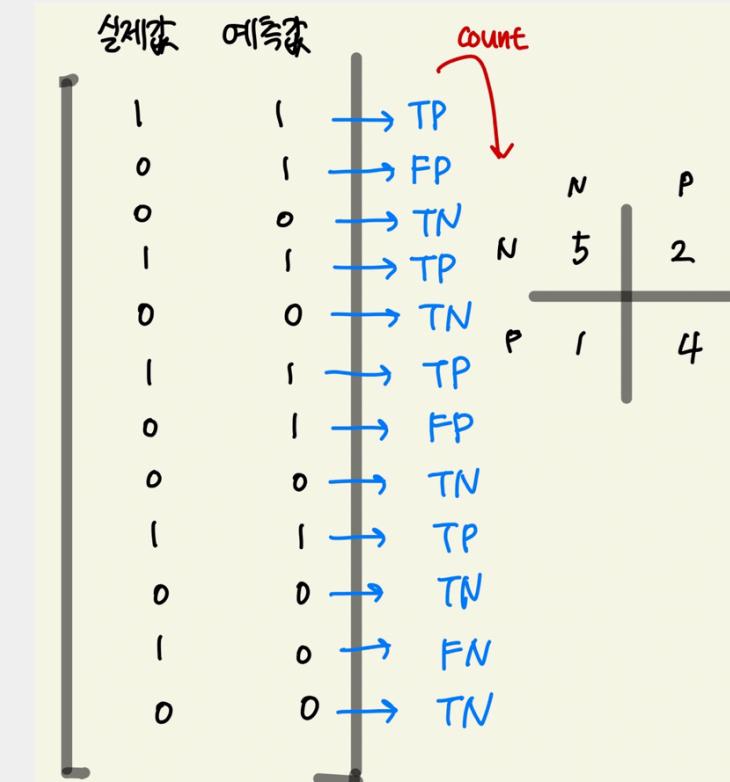
```
def get_clf_eval(y_test, pred):
    confusion = confusion_matrix(y_test, pred) #y_test와 예측값
    accuracy = accuracy_score(y_test, pred)
    precision = precision_score(y_test, pred)
get_clf_eval(y_test, pred) #get_clf_eval에 적용

data = {'y_Actual': [1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0],
        'y_Predicted': [1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0]}
df = pd.DataFrame(data, columns=['y_Actual','y_Predicted'])
print('df: \n', df)

df:
  y_Actual  y_Predicted
0         1             1
1         0             1
2         0             0
3         1             1
4         0             0
5         1             1
6         0             1
7         0             0
8         1             1
9         0             0
10        1             0
11        0             0

confusion = confusion_matrix(df['y_Actual'], df['y_Predicted'])
print(confusion)

[[5 2]
 [1 4]]
```



03 정밀도, 재현율

-정밀도/재현율 트레이드오프



Predict : Binarizer class

#Binarizer란?

```
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.]]
```

03 정밀도, 재현율

-정밀도/재현율 트레이드오프

Predict : Binarizer class(threshold=0.5)

#Binarizer의 threshold 설정값. 분류 결정 임곗값임.

```
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) #0.5를 기준으로

custom_predict = binarizer.transform(pred_proba_1) #pred_proba

```
get_clf_eval(y_test, custom_predict)
```

오차 행렬

```
[[104  14]
 [ 13  48]]
```

정확도: 0.8492, 정밀도: 0.7742, 재현율: 0.7869

```
lr_clf.fit(X_train, y_train)
```

```
pred = lr_clf.predict(X_test)
```

```
get_clf_eval(y_test, pred) #get_clf_eval에 적용
```

오차 행렬

```
[[104  14]
 [ 13  48]]
```

정확도: 0.8492, 정밀도: 0.7742, 재현율: 0.7869

```
....., .....,
```

```
[0.8883666, 0.1116334],
```

```
[0.20891303, 0.79108697],
```

```
[0.7829497, 0.2170503],
```

```
array([[1.],
```

```
[0.],
```

```
[0.],
```

```
[0.],
```

```
[0.],
```

```
[0.],
```

```
[0.],
```

```
[1.],
```

```
[0.],
```

03 정밀도, 재현율

-정밀도/재현율 트레이드오프

Predict : Binarizer class(threshold=0.4)

```
# Binarizer의 threshold 설정값을 0.4로 설정. 즉 분류 결정 임곗값을 0.5에서 0.4로 낮춤
custom_threshold = 0.4 #이거만 변경
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

```
[[98 20]
 [10 51]]
```

정확도: 0.8324, 정밀도: 0.7183, 재현율: 0.8361

03 정밀도, 재현율

-정밀도/재현율 트레이드오프

threshold=0.5 vs threshold=0.4

[임곗값 0.5일 때 오차 행렬]

TN	FP
108	10
FN	TP
14	47

[임곗값 0.4일 때 오차 행렬]

TN	FP
97	21
FN	TP
11	50

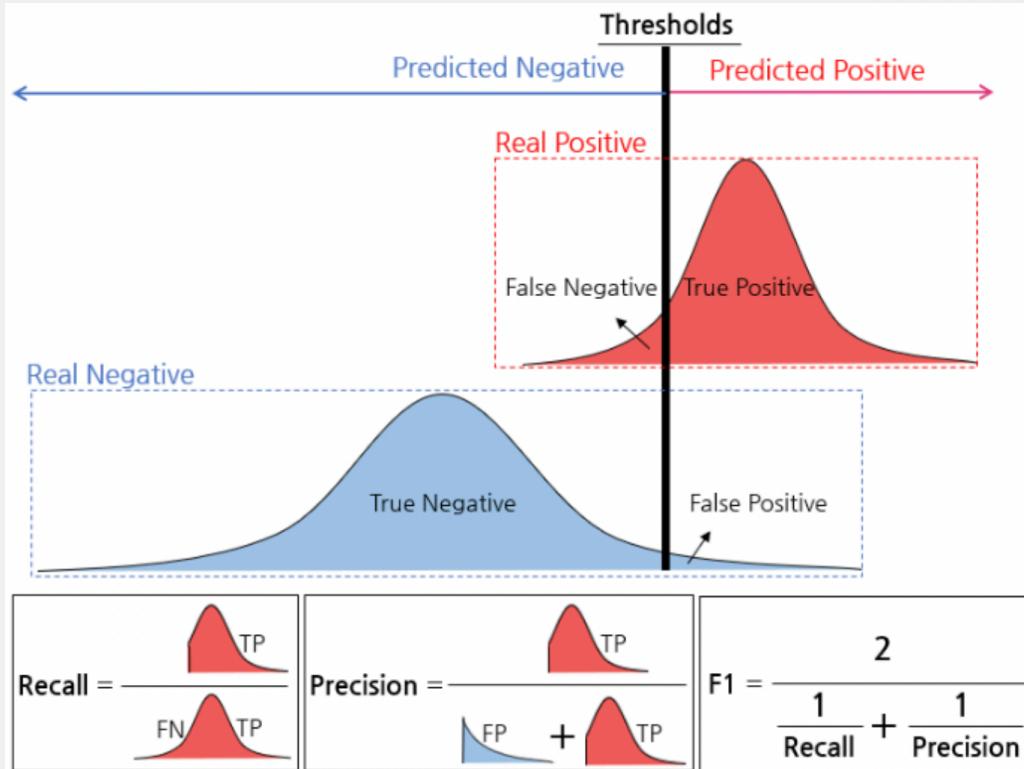
정확도: 0.8492, 정밀도: 0.7742, 재현율: 0.7869

정확도: 0.8324, 정밀도: 0.7183, 재현율: 0.8361

03 정밀도, 재현율

-정밀도/재현율 트레이드오프

Threshold 변경



03 정밀도, 재현율

-정밀도/재현율 트레이드오프

Threshold 조정하는 코드1

```
# 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 list 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)

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

평가 지표	분류 결정 임곗값				
	0.4	0.45	0.5	0.55	0.6
정확도	0.8212	0.8547	0.8659	0.8715	0.8771
정밀도	0.7042	0.7869	0.8246	0.8654	0.8980
재현율	0.8197	0.7869	0.7705	0.7377	0.7213

03 정밀도, 재현율

-정밀도/재현율 트레이드오프

Threshold 조정하는 코드2

```
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: (143,)
반환된 precisions 배열의 Shape: (144,)
반환된 recalls 배열의 Shape: (144,)
thresholds 5 sample: [0.10396968 0.10397196 0.10399758 0.10776186 0.10894507]
precisions 5 sample: [0.38853503 0.38461538 0.38709677 0.38961039 0.38562092]
recalls 5 sample: [1. 0.98360656 0.98360656 0.98360656 0.96721311]
샘플 추출을 위한 임계값 배열의 index 10개: [ 0 15 30 45 60 75 90 105 120 135]
샘플용 10개의 임곗값: [0.1 0.12 0.14 0.19 0.28 0.4 0.56 0.67 0.82 0.95]
샘플 임계값별 정밀도: [0.389 0.44 0.466 0.539 0.647 0.729 0.836 0.949 0.958 1. ]
샘플 임계값별 재현율: [1. 0.967 0.902 0.902 0.902 0.836 0.754 0.607 0.377 0.148]
```

thresholds

```
array([0.10396968, 0.10397196, 0.10399758, 0.10776186, 0.10894507,
       0.1116334, 0.11167113, 0.11168014, 0.11205761, 0.11209278,
       0.11625289, 0.11705654, 0.11735066, 0.11778907, 0.11806023,
       0.12138198, 0.12279917, 0.1228547, 0.12285576, 0.12286078,
       0.12437935, 0.12498449, 0.12550155, 0.1267594, 0.12842981,
       0.12887699, 0.13066155, 0.13285326, 0.13301961, 0.13308881,
       0.14034784, 0.14108319, 0.14113108, 0.14113822, 0.14466168,
       0.14544865, 0.1481038, 0.14810412, 0.14984461, 0.15344647,
       0.15427289, 0.16811186, 0.16852262, 0.18368732, 0.18518085,
       0.1862761, 0.19712382, 0.19840423, 0.19863853, 0.2170503,
       0.22632315, 0.23234988, 0.23368903, 0.23447311, 0.2400715,
       0.24639874, 0.25039798, 0.26657982, 0.27518341, 0.28010896,
       0.280406, 0.30445323, 0.30899618, 0.30932951, 0.31662165,
       0.32159357, 0.32192714, 0.34795346, 0.35055151, 0.35104858,
       0.35903183, 0.37999232, 0.38082728, 0.39928714, 0.40118679,
       0.4032477, 0.42849062, 0.43223425, 0.44301402, 0.44423628,
       0.44906573, 0.45378263, 0.48741378, 0.5016253, 0.50551088,
       0.51367075, 0.52999651, 0.53803543, 0.54494765, 0.54552456,
       0.56344821, 0.56983032, 0.59298726, 0.59419888, 0.61098482,
       0.62677689, 0.63046393, 0.63048051, 0.63068711, 0.63165631,
       0.63657885, 0.64533113, 0.65136317, 0.6537717, 0.6667874,
       0.66685611, 0.66767396, 0.67780213, 0.69044366, 0.70752724,
       0.72500523, 0.73507065, 0.74841371, 0.7514615, 0.77761097,
       0.79108697, 0.79293597, 0.79483139, 0.80338932, 0.81735033,
       0.82175583, 0.82605004, 0.8282423, 0.83556798, 0.85414617,
       0.87448448, 0.88093314, 0.88417086, 0.89296378, 0.90885084,
       0.9180886, 0.91898751, 0.9259316, 0.92884699, 0.93098179,
       0.94722454, 0.9480156, 0.94803994, 0.94882455, 0.94895635,
       0.9503597, 0.95182448, 0.96507068])
```

03 정밀도, 재현율

-정밀도/재현율 트레이드오프

Threshold 조정하는 코드3 (graph)

```
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker
%matplotlib inline

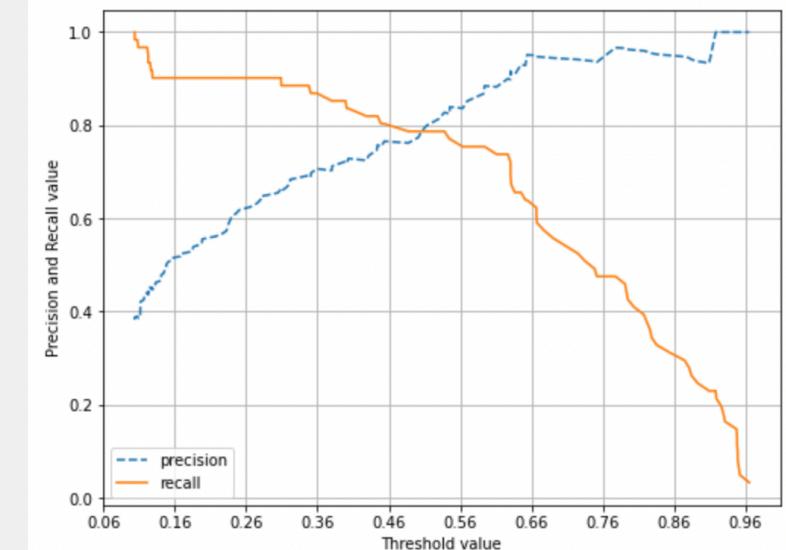
def precision_recall_curve_plot(y_test , pred_proba_c1):
    # threshold ndarray와 이 threshold에 따른 정밀도, 재현율 ndarray 추출.
    precisions, recalls, thresholds = precision_recall_curve( y_test, pred_proba_c1)

    # X축을 threshold값으로, Y축은 정밀도, 재현율 값으로 각각 Plot 수행. 정밀도는 점선으로 표시
    plt.figure(figsize=(8,6))
    threshold_boundary = thresholds.shape[0]
    plt.plot(thresholds, precisions[0:threshold_boundary], linestyle='--', label='precision')
    plt.plot(thresholds, recalls[0:threshold_boundary],label='recall')

    # threshold 값 X 축의 Scale을 0.1 단위로 변경
    start, end = plt.xlim() #plt.xlim()=0,1
    plt.xticks(np.round(np.arange(start, end, 0.1),2))

    # x축, y축 label과 legend, 그리고 grid 설정
    plt.xlabel('Threshold value'); plt.ylabel('Precision and Recall value')
    plt.legend(); plt.grid()
    plt.show()

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



CONTENTS

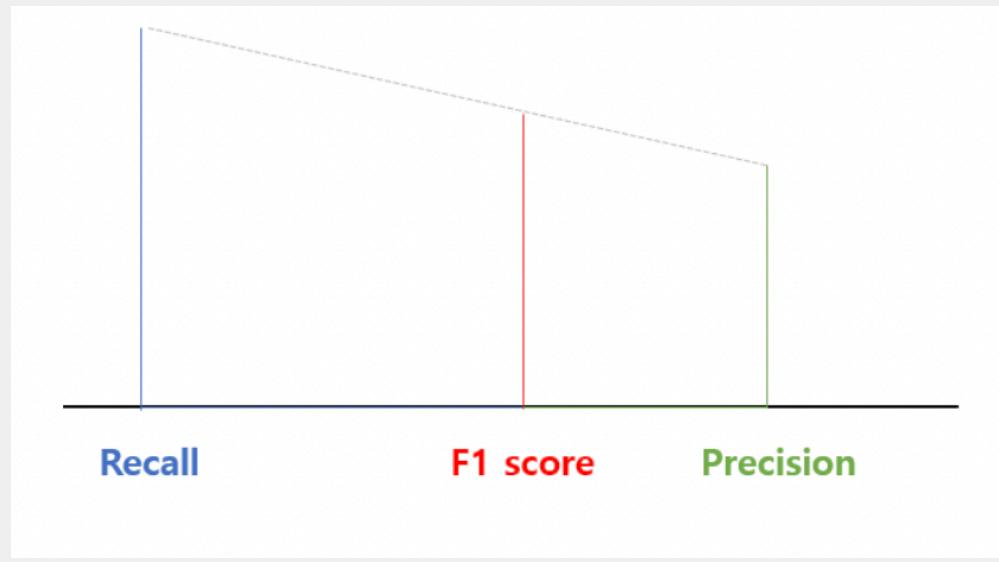
04. F1 Score

- F1 Score란?
- F1 Score = recall과 precision의 조화평균
- F1 Score 코드구현

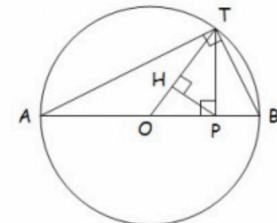
04 F1 Score

F1 Score = precision 과 recall의 조화평균

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



2. 산술,기하,조화평균의 기하학적 분석2



첫번째와 비슷합니다만 이번에는 원의 지름 AB 위의 점 P를 이용해보겠습니다.
PA = a, PB = b (단 a > b) 라고 했을 때 P에서 AB에 수직인 선이 원과 만나는 점을 T, P에서 OT에 내린 수선의 발을 H라고 하면

a,b의 산술평균(AP)은 원의 반지름 (즉 OT)
a,b의 기하평균(GP)은 PT
a,b의 조화평균(HP)은 TH

가 됩니다. 역시 식은 간단히 증명됩니다.
이에 의하여 역시

$$OT \geq PT \geq TH$$

조화평균을 쓴 이유는 precision과 recall이 0에 가까울 수록
F1 score도 동일하게 낮은 값을 갖게 하기 위해서.

04 F1 Score

F1 Score 구하는 코드 구현

```
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: {:.4f}, 정밀도: {:.4f}, 재현율: {:.4f}, F1:{:.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

```
[[98 20]
 [10 51]]
```

정확도: 0.8324, 정밀도: 0.7183, 재현율: 0.8361, F1:0.7727

CONTENTS

05. ROC, AUC

- TPR, FPR이란
- ROC 곡선 위의 점 의미
- ROC 곡선의 흰 정도 의미
- 코드 구현

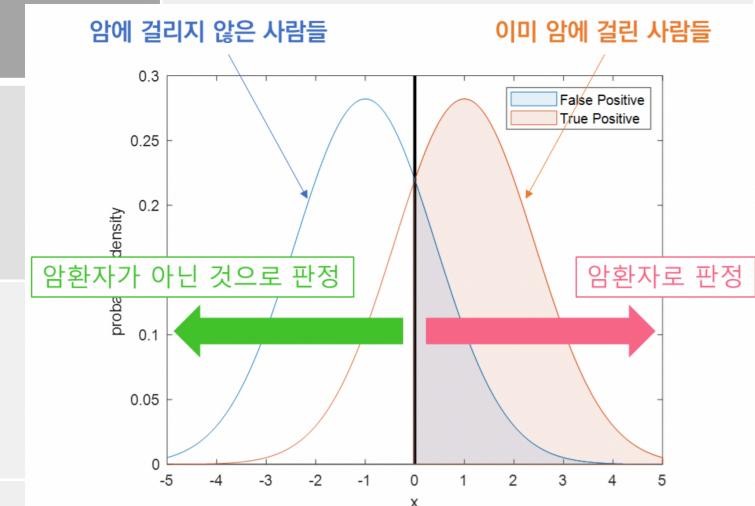
05 ROC, AUC

TPR, FPR

$$TPR = \frac{TP}{all\ positive} = \frac{TP}{TP+FN}$$

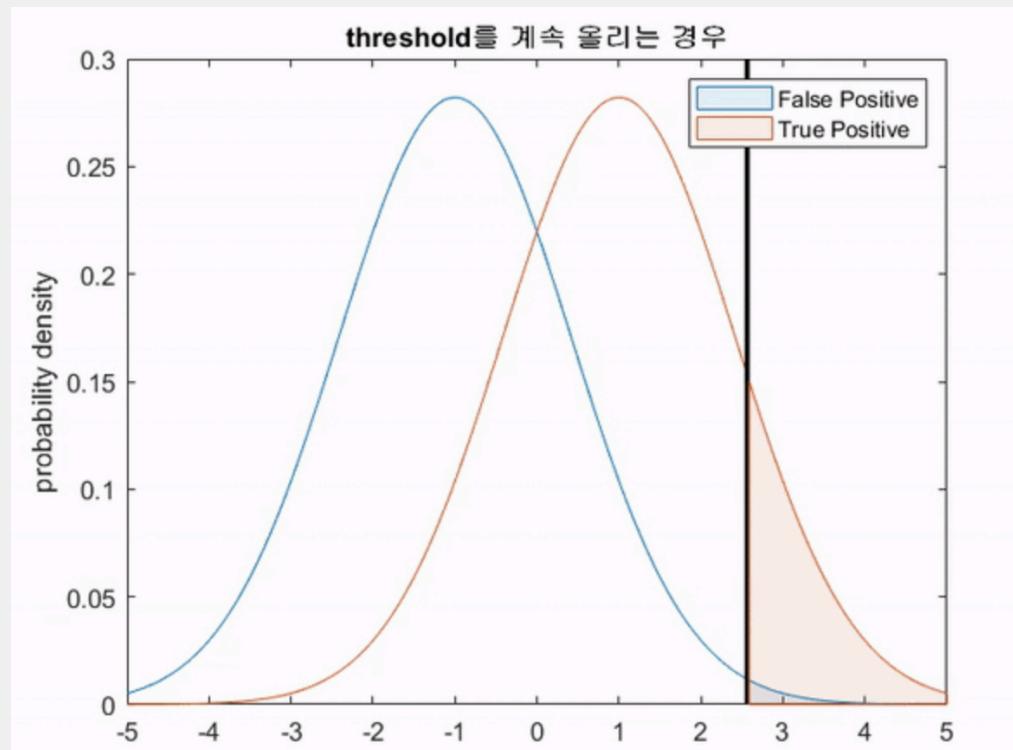
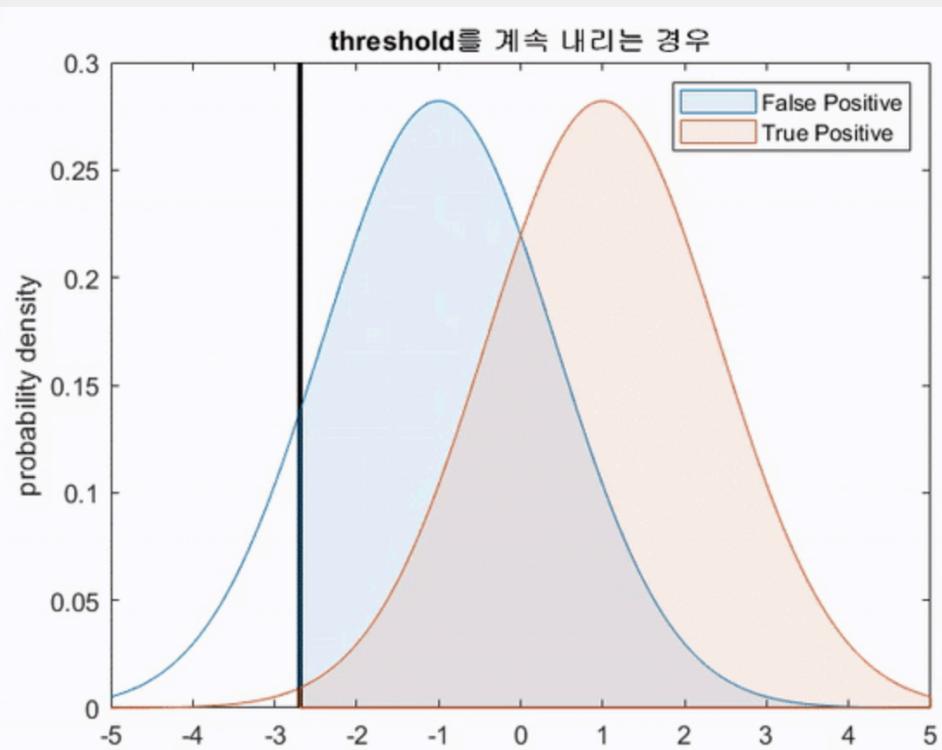
$$FPR = \frac{FP}{all\ negative} = \frac{FP}{FP+TN}$$

	Negative (암환자 아니라 예측)	Positive (암환자라 예측)
False (암환자 아닌 경우)	TN	FP
True (암환자인 경우)	FN	TP



05 ROC, AUC

Threshold에 따른 FPR, TPR



05 ROC, AUC

ROC 곡선 위의 점 의미

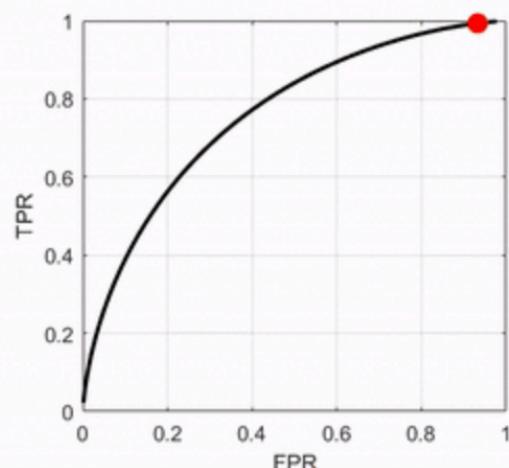
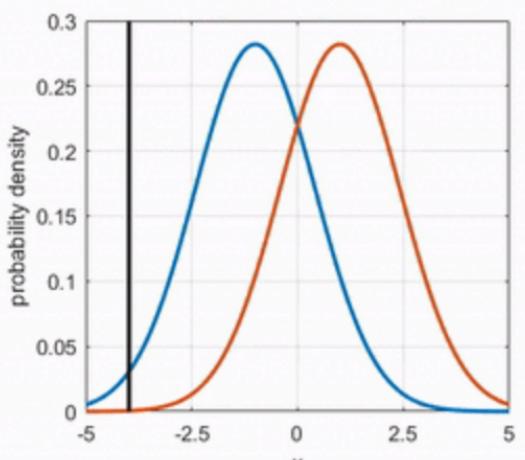


그림 6. threshold 변화에 따른 ROC 커브 위의 점 위치 변화

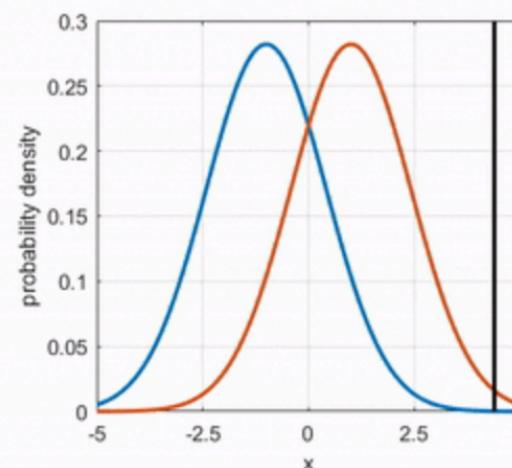
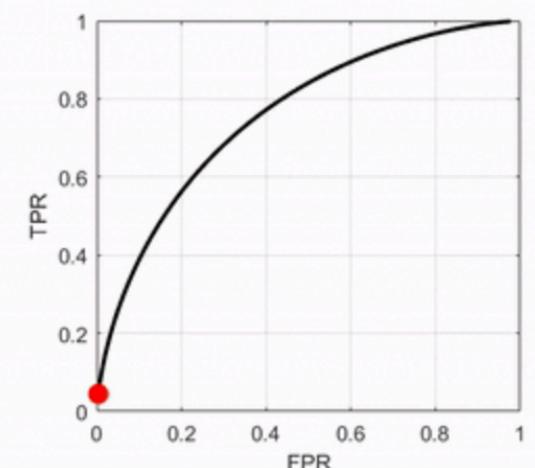


그림 6. threshold 변화에 따른 ROC 커브 위의 점 위치 변화



05 ROC, AUC

ROC 곡선의 흰 정도가 의미하는 것은?

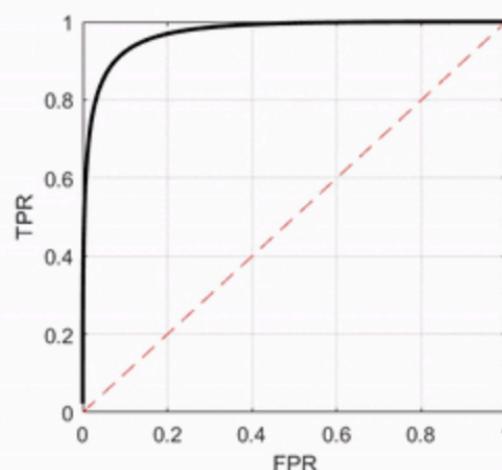
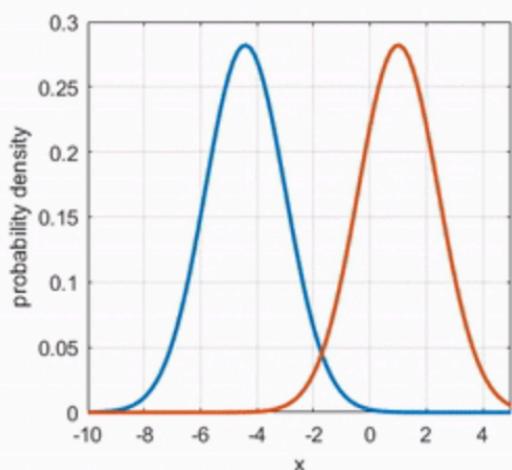


그림 7. 두 그룹을 더 잘 구별할 수 있을수록 ROC 커브는 좌상단에 붙게 된다.

AUC가 높다(1에 가까움)

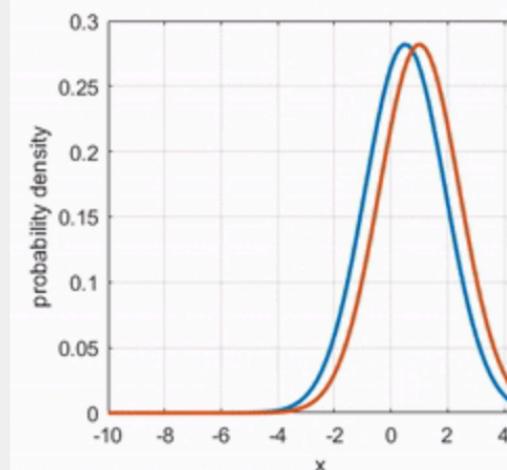


그림 7. 두 그룹을 더 잘 구별할 수 있을수록 ROC 커브는 좌상단에 붙게 된다.

AUC가 낮다(0.5에 가까움)

05 ROC, AUC

F1 Score 구하는 코드 구현

```
from sklearn.metrics import roc_curve

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

fprs, tprs, thresholds = roc_curve(y_test, pred_proba_class1)
print('thresholds[0]:', thresholds[0])
# 반환된 임곗값 배열로 우가 47건이므로 샘플로 10건만 추출하되, 임곗값을 5 Step으로 추출.
thr_index = np.arange(0, thresholds.shape[0], 5)
print('샘플 추출을 위한 임곗값 배열의 index 10개:', thr_index)
print('샘플용 10개의 임곗값: ', np.round(thresholds[thr_index], 2))

# 5 step 단위로 추출된 임곗값에 따른 FPR, TPR 값
print('샘플 임곗값별 FPR: ', np.round(fprs[thr_index], 3))
print('샘플 임곗값별 TPR: ', np.round(tprs[thr_index], 3))

max predict_proba: 0.9650706802232637
thresholds[0]: 1.9650706802232638
샘플 추출을 위한 임곗값 배열의 index 10개: [ 0  5 10 15 20 25 30 35 40 45 50]
샘플용 10개의 임곗값:  [1.97 0.75 0.63 0.59 0.49 0.4  0.35 0.23 0.13 0.12 0.11]
샘플 임곗값별 FPR:  [0.      0.017 0.034 0.051 0.127 0.161 0.203 0.331 0.585 0.636 0.797]
샘플 임곗값별 TPR:  [0.      0.475 0.689 0.754 0.787 0.836 0.869 0.902 0.918 0.967 0.967]
```

```
from sklearn.metrics import roc_auc_score

### 아래는 roc_auc_score()의 인자를 잘못 입력한 것으로, 책에서 수정이 필요한 부분입니다.
### 책에서는 roc_auc_score(y_test, pred)로 예측 타겟값을 입력하였으나
### roc_auc_score(y_test, y_score)로 y_score는 predict_proba()로 호출된 예측
#pred = lr_clf.predict(X_test)
#roc_score = roc_auc_score(y_test, pred)

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

05 ROC, AUC

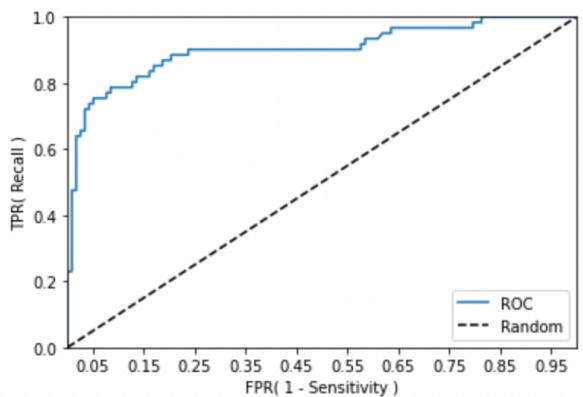
ROC 곡선 graph 코드 구현

```
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] )
```



CONTENTS

06. 피마 인디언 당뇨병 예측

- 어떤 성능 지표를 보정해야 할까?
- 전처리 후 달라진 성능 평가 지표
- Threshold 변경 후 성능 평가 지표

06 피마 인디언 당뇨병 예측

어떤 성능 평가 지표 보정?

```
# 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.

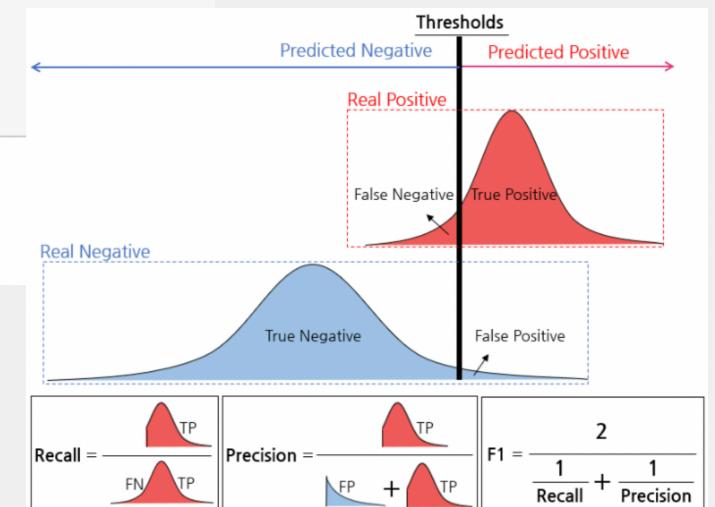
```
lr_clf = LogisticRegression()  
lr_clf.fit(X_train, y_train)  
pred = lr_clf.predict(X_test)  
pred_proba = lr_clf.predict_proba(X_test)[:, 1]  
  
get_clf_eval(y_test, pred, pred_proba)
```

오차 행렬

```
[[88 12]  
 [23 31]]
```

정확도: 0.7727, 정밀도: 0.7209, 재현율: 0.5741, F1: 0.6392, AUC: 0.7919

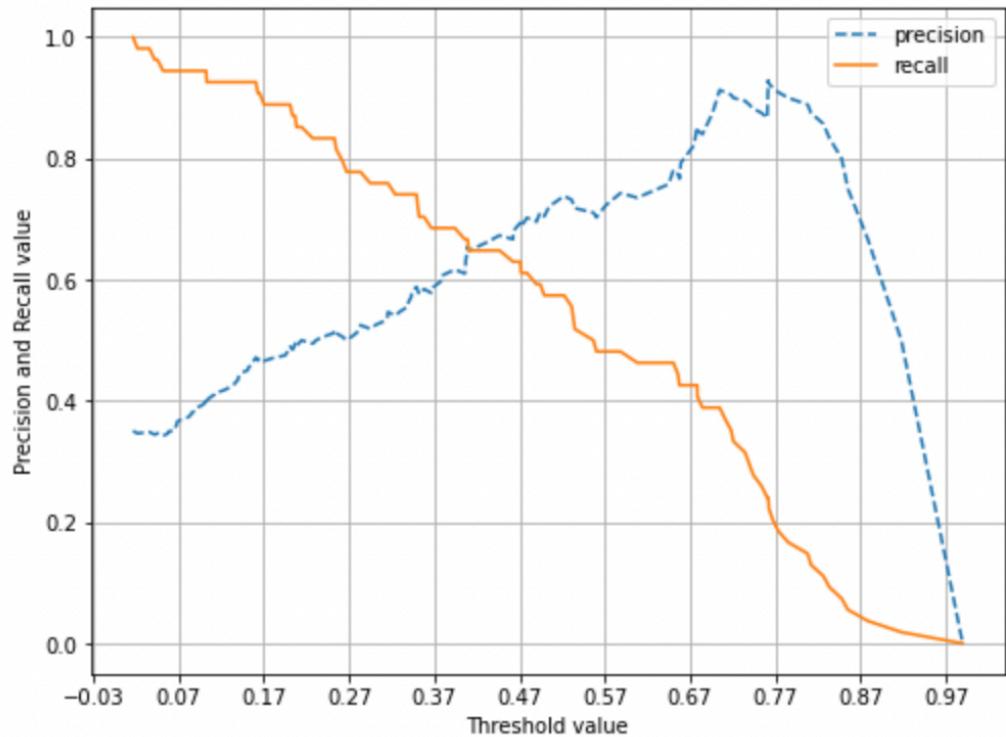
예측 정확도가 77.27%, 재현율은 59.26%로 측정됐습니다. 전체 데이터의 65%가 Negative이므로 정확도보다는 재현율 성능에 조금 더 초점을 맞춰 보겠습니다. 먼저 정밀도 재현율 곡선을 보고 임계값별 정밀도와 재현율 값의 변화를 확인하겠습니다. 이를 위해 precision_recall_curve_plot() 함수를 이용하겠습니다.



06 피마 인디언 당뇨병 예측

Precision recall curve -> 균형점

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



06 피마 인디언 당뇨병 예측

전처리

```
diabetes_data.describe()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	0.471876	33.240885	0.348958
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.331329	11.760232	0.476951
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.000000	0.000000
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.243750	24.000000	0.000000
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0.372500	29.000000	0.000000
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250	41.000000	1.000000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000	1.000000

06 피마 인디언 당뇨병 예측

전처리 후 성능 평가 지표

```
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

06 피마 인디언 당뇨병 예측

Threshold 변경

```
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)
```

평가 지표	분류 결정 임곗값							
	0.3	0.33	0.36	0.39	0.42	0.45	0.48	0.50
정확도	0.7013	0.7403	0.7468	0.7532	0.7792	0.7857	0.7987	0.7987
정밀도	0.5513	0.5972	0.6190	0.6333	0.6923	0.7059	0.7447	0.7674
재현율	0.7963	0.7963	0.7222	0.7037	0.6667	0.6667	0.6481	0.6111
F1	0.6515	0.6825	0.6667	0.6667	0.6792	0.6857	0.6931	0.6804
ROC AUC	0.7231	0.7531	0.7411	0.7419	0.7533	0.7583	0.7641	0.7556

06 피마 인디언 당뇨병 예측

최종 결과

```
# 임곗값을 0.48로 설정한 Binarizer 생성
binarizer = Binarizer(threshold=0.48)

# 위에서 구한 lr_clf의 predict_proba() 예측 확률 array에서 1에 해당하는 컬럼값을 Binarizer변환.
pred_th_048 = binarizer.fit_transform(pred_proba[:, 1].reshape(-1,1))

get_clf_eval(y_test , pred_th_048, pred_proba[:, 1])
```

오차 행렬

```
[[88 12]
 [19 35]]
```

정확도: 0.7987, 정밀도: 0.7447, 재현율: 0.6481, F1: 0.6931, AUC:0.8433

감사합니다

