비타민 8기 3조 곽아람, 김보현, 김진호, 홍성현

평가(Evaluation)

CONTENTS

1. 데이터 분석 순서

데이터 분석 순서

머신러닝의 프로세스

stage 1	stage 2	stage 3	stage 4	stage 5	stage 6
Domain Understanding And Data Collection	Data Preprocessing	Modeling And Ensemble	Prediction	Evaluation	Deployment

성능 평가 지표의 유형

모델
- 분류
  - 정확도(Accuracy)
  - 오차행렬(Confusion Matrix)
  - 정밀도(Precision)
  - 재현율(Recall)
  - F1 스코어
  - ROC AUC
- 회귀

※ 테스트 데이터는 모델링 과정에서 사용 금지!

-> 모델이 새로운 데이터에 대해 얼마나 일반화 가능한지 측정하기 위해 **(평가의 목적)**

2. 머신러닝 모델의 성능 평가

정확도(Accuracy)

정확도(Accuracy)란?

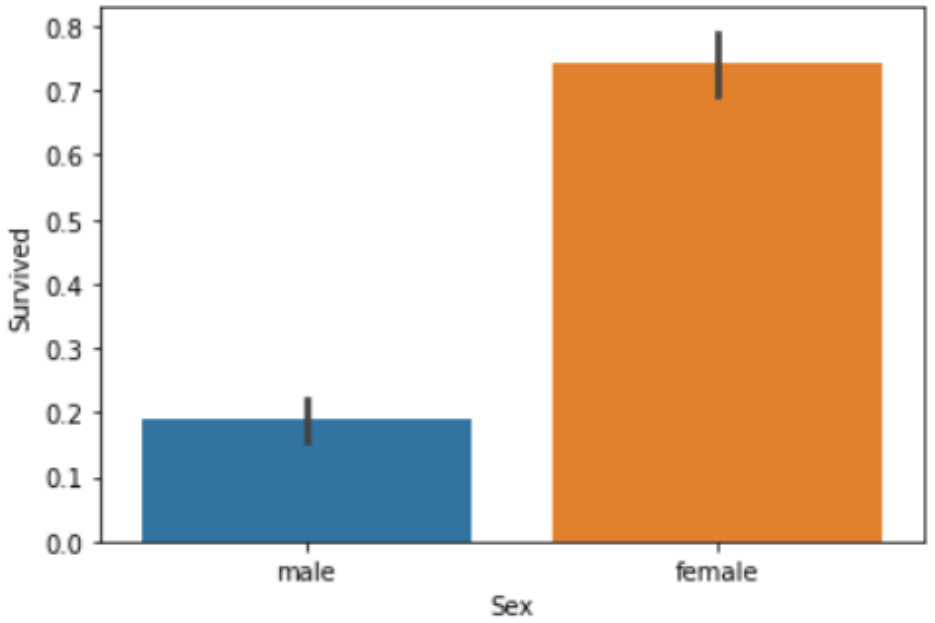
: 실제 데이터에서 예측 데이터가 얼마나 같은지를 판단하는 지표 (직관적)

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

이진 분류의 경우 데이터의 구성에 따라 모델의 성능을 왜곡할 수 있기 때문에 정확도 수치 하나로만 판단 X

타이타닉 생존자 예측

```
import pandas as pd
import numpy as np
import seaborn as sns

titanic = pd.read_csv('titanic_train.csv')
titanic.head()
```



1. 사이킷런의 BaseEstimator 클래스를 상속받아 Customized 형태의 Estimator 생성

```
from sklearn.base import BaseEstimator

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

    # predict() 메서드는 단순이 Sex 피쳐가 1이면 0, 아니면 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
```

> MyDummyClassifier 클래스

fit() -> 아무것도 수행하지 않음

predict() -> 'Sex'=1 이면 0, else 1 로 예측

정확도(Accuracy)

2. transform_features() 함수 지정

```
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 속성 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 Data Preprocessing 함수 호출
def transform_features(df):
    df = fillna(df)
    df = drop_features(df)
    df = format_features(df)
    return df
```

3. 데이터 가공, 학습/테스트 데이터 분할

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# 데이터 가공, 학습/테스트 데이터 분할
y_titanic = titanic['Survived']
X_titanic = titanic.drop('Survived', axis = 1)
X_titanic = transform_features(X_titanic)
X_train, X_test, y_train, y_test = train_test_split(X_titanic, y_titanic,
                                                    test_size = 0.2, random_state = 0)
```

4. MyDummyClassifier()를 이용해 생존자 예측

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

my_prediction = my_clf.predict(X_test)
print('MyDummyClassifier의 정확도는 :{0:.4f}'.format(accuracy_score(y_test, my_prediction)))
```

```
MyDummyClassifier의 정확도는 :0.7877
```

단순한 알고리즘으로 예측을 하더라도 불균형한 레이블 값 분포에서 ML 모델의 성능을 판단할 경우, 적합한 평가 지표가 아니다!!

MNIST

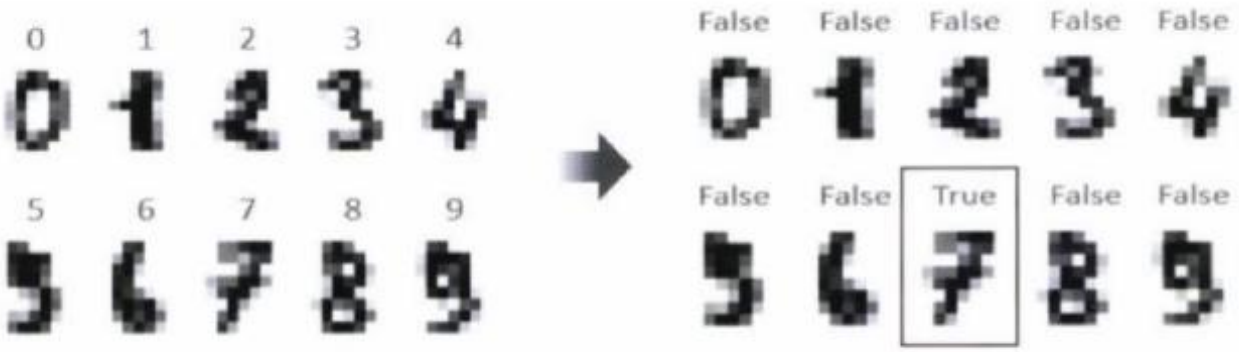
load_digits() API통해 제공

: 0~9까지의 숫자 이미지의 픽셀 정보를 가짐 / 숫자 Digit 예측에 사용

왜 불균형한 레이블 값 분포에서 정확도는
적합한 평가지표가 아닐까?

100개의 데이터 중 90개의 데이터 레이블 = 0,
10개의 데이터 레이블 = 1 이라고 한다면
0으로 예측 결과를 반환하는 ML 모델의 경우
정확도는 90%가 되기 때문

MNIST에서 멀티 분류를 이진분류로 변경



레이블 값이 7인 것만 True, 나머지 값은 False
-> 전체 데이터의 10%만 True, 90%는 False

아무것도 하지 않고 특정한 결과로 설정하여도
데이터 분포도가 균일하지 않은 경우
높은 정확도 수치를 보임

MNIST 데이터 세트

1. MyFakeClassifier() 클래스 생성

```
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 0 값으로 만들어서 반환
    def predict(sel, X):
        return np.zeros((len(X),1), dtype = bool)
```

2. MNIST 데이터 로딩 및 변환

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

# digits 번호가 7번이면 True이고 이를 astype(int)로 1로 변환, else False, 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)
```

3. 정확도 판단

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

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

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

> MyFakeClassifier() 클래스
>
> fit() -> 아무것도 수행하지 않음
>
> 모든 예측을 0으로 하여도 정확도 = 90%

오차 행렬(Confusion Matrix)

오차 행렬(Confusion Matrix)이란?

: 이진 분류의 예측 오류가 얼마인지와 더불어 어떠한 유형의 예측 오류가 발생하고 있는지 함께 나타내는 지표

		예측 클래스 (Predicted Class)	
		Nagative(0)	Positive(1)
실제 클래스 (Actual Class)	Nagative(0)	TN (True Negative)	FP (False Positive)
	Positive(1)	FN (False Negative)	TP (True Positive)

- TN : 예측값 = Negative(0)
  실제값 = Negative(0)
- FN : 예측값 = Negative(0)
  실제값 = Positive(1)
- FP : 예측값 = Positive(1)
  실제값 = Negative(0)
- TP : 예측값 = Positive(1)
  실제값 = Positive(1)

1, 2종 오류

	귀무가설이 참 (불이 나지 않았음)	귀무가설이 거짓 (화재 발생)
귀무가설 기각하지 않음 (화재 경보 울리지 않음)	옳은 결정	2종 오류: Miss (불이 났는데도 화재 경보 울리지 않음)
귀무가설 기각 (화재 경보 울림)	1종 오류: False Alarm (불이 나지 않았는데도 화재 경보 울림)	옳은 결정

- 1종 오류

귀무가설이 참인데 잘못 판단해 기각 해버리는 오류

-> FN를 낮춰야함!!

- 2종 오류

귀무가설을 거짓인데도 기각하지 않아서 생기는 오류

-> FP를 낮춰야함!!

오차행렬(Confusion Matrix)

1. confusion_matrix()로 배열 형태 출력

```
from sklearn.metrics import confusion_matrix

confusion_matrix(y_test, fake_pred)
```

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

오차행렬(Confusion Matrix)

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

(예측 결과와 실제 값이 동일한 건수)

(전체 예측 데이터 건수)

불균형한 레이블 클래스를 가지는 이진 분류 모델	중점적으로 찾아야하는 매우 적은 수의 결과값 – Positive
	그렇지 않은 경우 – Negative
Ex1) 사기 행위 예측 모델	사기 행위 – Positive
	정상 행위 – Negetive
Ex2) 암 검진 예측 모델	양성 (암 O) – Positive
	음성 (암 X) – Negative

오차행렬(Confusion Matrix)

불균형한 이진분류 데이터 세트에서는 Positive 데이터 건수가 매우 작기 때문에 데이터에 기반한 ML알고리즘은 Positive보다 **Negative로 예측 정확도가 높아지는 경향** 발생

10000건의 데이터 세트	9900건	Negative
	100건	Positive

- Negative로 예측하는 경향이 더 강해짐 -> TN 매우 커짐. TP 매우 작아짐
- Negative로 예측할 때 정확도가 높아짐 -> FN 매우 작아짐
- Positive로 예측하는 경우가 작음 -> FP 매우 작아짐

정확도 지표는 비대칭한 데이터 세트에서 **Positive**에 대한 예측 정확도를 판단하지 못해 Negative에 대한 예측 정확도만으로도 **분류의 정확도가 매우 높게 나타나는** 수치적인 판단 오류를 일으킴

불균형 데이터 세트에서 정밀도(Precision)와 재현율(Recall)을 사용한다!!

정밀도(Precision)와 재현율(Recall)이란?

: Positive 데이터 세트의 예측 성능에 좀 더 초점을 맞춘 평가 지표

$$\text{정밀도} = \frac{TP}{FP + TP}$$

(= 양성 예측도)

(예측 결과와 실제 값이 동일한 건수)

(예측을 Positive로 한 건수)

$$\text{재현율} = \frac{TP}{FN + TP}$$

(= 민감도, TPR)

(예측 결과와 실제 값이 동일한 건수)

(실제 값이 Positive인 건수)

정밀도와 재현율

> 이진 분류 모델의 업무 특성에 따른 지표

정밀도(Precision)

실제 Negative인 데이터를 Positive로 잘 못 판단하면 업무상 큰 영향을 입는 경우

$$\frac{TP}{\downarrow FP + TP}$$

Ex) 스팸메일 여부 판단 모델

실제 Negative인 일반 메일을 Positive인 스펨 메일로 분류할 경우 이메일을 받지 못하게 되어 업무에 차질 발생

재현율(Recall)

실제 Positive인 데이터를 Negative로 잘 못 판단하면 업무상 큰 영향을 입는 경우

$$\frac{TP}{\downarrow FN + TP}$$

Ex 1) 암 판단 모델

실제 Positive인 환자를 Negative로 판단할 경우 생명을 앗아갈 위험 발생

Ex 2) 금융 사기 적발 모델

실제 금융거래 사기인 Positive 건을 Negative로 판단하면 회사에 손해 발생

정밀도와 재현율

1. 오차행렬, 정확도, 정밀도, 재현율 계산하는 함수 생성

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

# 오차 행렬, 정확도, 정밀도, 재현율을 한꺼번에 계산하는 함수 생성
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 회귀란?

선형 회귀 방식을 분류에 적용한 알고리즘

2. 로지스틱 회귀 기반으로 타이타닉 생존자 예측

```
## 로지스틱 회귀 기반으로 타이타닉 생존자 예측 후 위의 평가 진행
from sklearn.linear_model import LogisticRegression

# 원본 데이터를 재로딩, 데이터 가공, 학습/테스트 데이터 분할
titanic = pd.read_csv('titanic_train.csv')
y_titanic = titanic['Survived']
X_titanic = titanic.drop('Survived', axis = 1)
X_titanic = transform_features(X_titanic)

X_train, X_test, y_train, y_test = train_test_split(X_titanic, y_titanic,
                                                    test_size = 0.2, 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)
```

정밀도 < 재현율

재현율 또는 정밀도를 좀 더 강화할 방법은 무엇일까?

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

정밀도/재현율 트레이드 오프

: 분류하려는 업무의 특성상 정밀도 또는 재현율이 특별이 강조돼야 할 경우 분류의 결정 임곗값(Threshold)을 조정해 정밀도 또는 재현율 수치를 높일 수 있다.

정밀도 ↑ → 재현율 ↓

재현율 ↑ → 정밀도 ↓

		예측 클래스	
		Negative	Positive
실제 클래스	Negative	TN	FP
	Positive	FN	TP

정밀도와 재현율 – 트레이드 오프

1. predict_proba() 메서드 사용

```
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])
print('\n')

# 예측 확률 array와 예측 결과값 array를 concatenate 하여 예측 확률과 결과값을 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
: [[0.46170212 0.53829788]
 [0.87864222 0.12135778]
 [0.87728507 0.12271493]]
```

두 번째 칼럼 : 클래스 값 1에 대한 예측 확률

첫 번째 칼럼 : 클래스 값 0에 대한 예측 확률

첫 번째 칼럼 + 두 번째 칼럼 = 1

```
두개의 class 중에서 더 큰 확률을 클래스 값으로 예측
 [[0.46170212 0.53829788 1.        ]
 [0.87864222 0.12135778 0.        ]
 [0.87728507 0.12271493 0.        ]]
```

predict_porba()
개별 클래스 예측 확률

predict()
두 개 칼럼 중 더 큰 확률값으로 최종 예측

predict_proba()

테스트 피처 데이터 세트입력

테스트 피처 레코드의 개별 클래스 예측 확률 반환

임계값 = 50%보다 크면 Positive
작거나 같으면 Negative로 결정

Predict_proba() 호출 결과로 반환된 배열에서 분류 결정 임계값 보다 큰 값이 들어 있는 칼럼의 위치를 받아서 최종적으로 예측 클래스를 결정

정밀도와 재현율 – 트레이드 오프

2. Binarizer 클래스 사용

```
from sklearn.preprocessing import Binarizer

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

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

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

Ndarray의 값이 지정된 threshold(1.1)
보다 같거나 작으면 0, 크면 1로 변환

3. 임계값 0.5로 지정 후 예측 수행

```
from sklearn.preprocessing import Binarizer

# Binarizer의 threshold 설정값, 분류 결정 임계값
custom_threshold = 0.5

# predict_proba() 반환값의 두번째 칼럼
# 즉 Positive 클래스 칼럼 하나만 추출하여 Binarizer 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
[[104  14]
 [ 13  48]]
정확도: 0.8492, 정밀도: 0.7742, 재현율: 0.7869
```

앞서 predict()로 계산된 지표와 값이 같다.
즉, predict()는 predict_proba()에 기반

정밀도와 재현율 – 트레이드 오프

4. 임계값 0.4로 수정 후 예측 수행

```
## 임계값 = 0.4

custom_threshold = 0.4
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
[[99 19]
 [10 51]]
정확도: 0.8380, 정밀도: 0.7286, 재현율: 0.8361
```

임계값이 0.5일 때보다 정밀도는 떨어지고 재현율은 올라갔음을 알 수 있다.

왜 임계값의 변화에 따라 재현율과 정밀도의 값이 변할까?

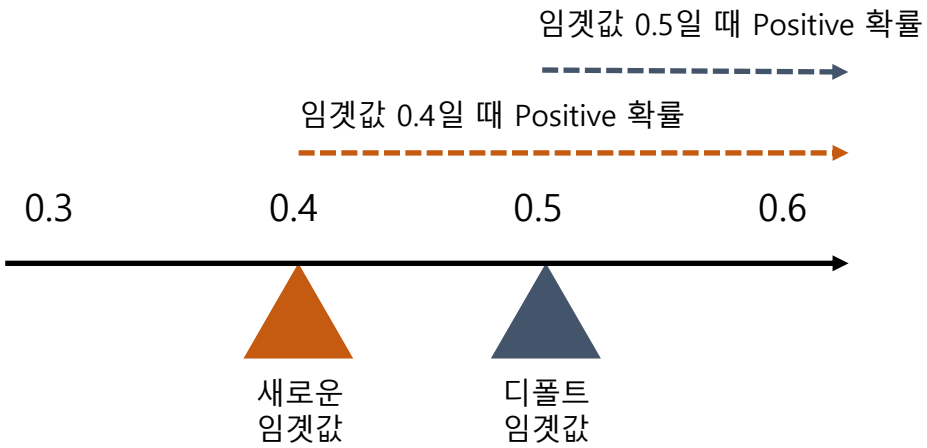
정밀도와 재현율 – 트레이드 오프

분류 결정 임계값은 Positive 예측값을 결정하는 확률의 기준

분류 결정 임곗값이 낮아질수록 Positive로 예측할 확률이 높아짐 -> 재현율 증가

임계값↓ → 재현율↑ 정밀도↓

(positive값) TP/(FN+TP) TP/(FP+TP)



Positive로 예측 ↓

실제 positive →	TN 104	FP 14
	FN 13	TP 48

임계값 0.5일 때 오차 행렬

음성 예측↓	TN 99	FP 19
	FN 10	TP 51

양성 예측↑

임계값 0.4일 때 오차 행렬

정밀도와 재현율 – 트레이드 오프

5. 임계값 리스트화 후 예측 수행

```
# 테스트를 수행할 모든 임계값 리스트화
thresholds = [0.4, 0.45, 0.5, 0.55, 0.6]

def get_eval_by_threshold(y_test, pred_proba_c1, thresholds):
    # thresholds list 객체내의 값을 차례로 iteration 하면서 Evaluation 수행
    for custom_threshold in thresholds:
        binarizer = Binarizer(threshold = custom_threshold).fit(pred_proba_c1)
        custom_predict = binarizer.transform(pred_proba_c1)
        print('임계값 : ', custom_threshold)
        get_clf_eval(y_test, custom_predict)
        print('')

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

```
임계값 :  0.4
오차 행렬
[[99 19]
 [10 51]]
정확도: 0.8380, 정밀도: 0.7286, 재현율: 0.8361

임계값 :  0.45
오차 행렬
[[103  15]
 [ 12  49]]
정확도: 0.8492, 정밀도: 0.7656, 재현율: 0.8033

임계값 :  0.5
오차 행렬
[[104  14]
 [ 13  48]]
정확도: 0.8492, 정밀도: 0.7742, 재현율: 0.7869

임계값 :  0.55
오차 행렬
[[109   9]
 [ 15  46]]
정확도: 0.8659, 정밀도: 0.8364, 재현율: 0.7541

임계값 :  0.6
오차 행렬
[[112   6]
 [ 16  45]]
정확도: 0.8771, 정밀도: 0.8824, 재현율: 0.7377
```

5. 임계값 리스트화 후 예측 수행

평가 지표	분류 결정 임계값				
	0.4	0.45	0.5	0.55	0.6
정확도	0.8380	0.8492	0.8492	0.8659	0.8771
정밀도	0.7286	0.7656	0.7742	0.8364	0.8824
재현율	0.8361	0.8033	0.7869	0.7541	0.7377

6. 타이타닉 예측 모델의 입계값 별 정밀도, 재현율

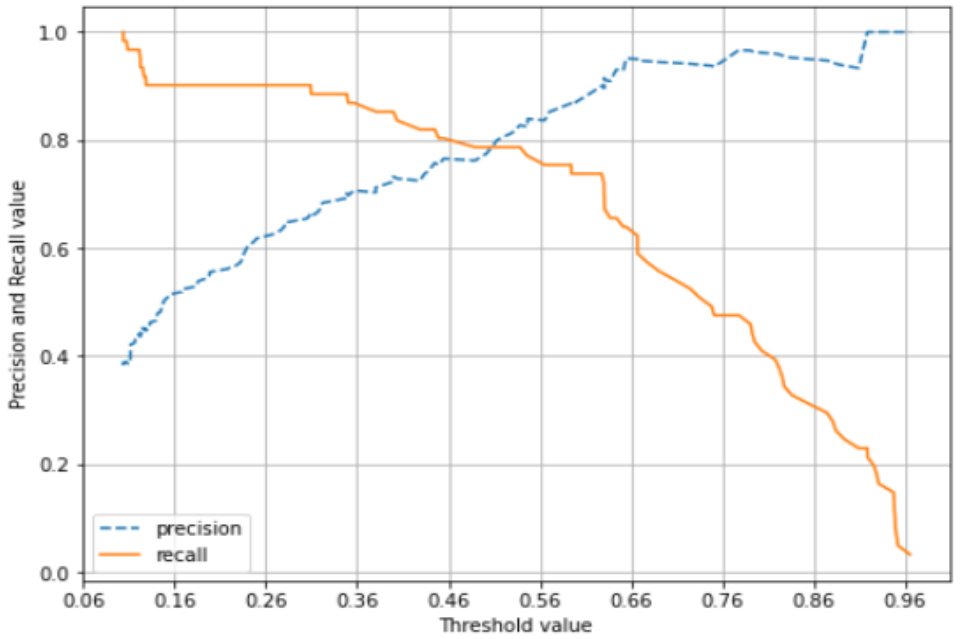
```
from sklearn.metrics import precision_recall_curve

# 레이블 값이 1 일때의 예측 확률 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 정밀도 배열의 Shape:', precisions.shape)
print('반환된 재현율 배열의 Shape:', recalls.shape)
print('')

print("임계값 5 sample:", thresholds[:5])
print("정밀도 5 sample:", precisions[:5])
print("재현율 5 sample:", recalls[:5])
print('')

# 반환된 임계값 배열 로우가 143건이므로 샘플로 10건만 추출, 임계값을 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))
print('')

# 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 정밀도 배열의 Shape: (144,)
반환된 재현율 배열의 Shape: (144,)

임계값 5 sample: [0.10392303 0.10392523 0.10394993 0.10734518 0.10890452]
정밀도 5 sample: [0.38853503 0.38461538 0.38709677 0.38961039 0.38562092]
재현율 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  0.57 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]
```

정밀도와 재현율 – 트레이드 오프

7. 임계값 별 정밀도, 재현율 곡선으로 표현

```
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
    precision, recalls, thresholds = precision_recall_curve(y_test, pred_proba_c1)

    # X축을 threshold값으로, Y축을 정밀도, 재현율 값으로 각각 Plot 수행(정밀도=점선)
    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 0.1단위로 변경
    start, end = plt.xlim()
    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] )
```



정밀도와 재현율 – 정밀도의 재현율의 맹점

Positive 예측 임곗값을 변경함에 따라 정밀도와 재현율의 수치가 변경

임곗값의 이러한 변경은 업무 환경에 맞게 두 개의 수치를 상호 보완할 수 있는 수준에서 적용되어야 함

정밀도가 100%가 되는 방법	재현율이 100%가 되는 방법
확실한 기준이 되는 경우만 Positive로 예측하고 나머지는 모두 Negative로 예측 Ex) 전체 환자 1000명 중 확실한 Positive 징후만 가진 환자는 단 1명이라고 하면 이 한명만 Positive로 예측하고 나머지는 모두 Negative로 예측 정밀도 = TP / (FP+TP) = 1 / (1+0) = 100%	모두 Positive로 예측 Ex) 전체 환자 1000명을 모두 Positive로 예측 실제 양성인 사람이 30명 정도라도 TN이 수치에 포함되지 않고 FN 은 아예 0이므로 재현율 = TP / (FN + TP) = 30 / (0 + 30) = 100%

정밀도/재현율 중 하나만 스코어가 좋고 다른 하나는 스코어가 나쁜 분류는 성능이 좋지 않은 분류로 간주

-> 정밀도와 재현율이 어느 한쪽으로 치우치지 않는 수치를 나타내는 F1 스코어를 활용

F1 SCORE

F1 score 란?

앞서 설명한 precision과 recall은 모델을 이해하기 위한 유용한 지표이지만, 우리는 모델이 얼마나 효과적인지를 설명하는 지표가 필요한데 이것이 F1 score입니다.

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

F1 score는 precision과 recall의 조화평균입니다.

	Precision	recall	F1 score
모델A	1	0.1	0.18
모델B	0.5	0.5	0.5

F1 score 사용 방법

다행히 앞서 설명한 계산식을 우리가 직접 계산할 필요는 없습니다.
사이킷런 f1_score API 제공해 줍니다.

```
from sklearn.metrics import f1_score

f1 = f1_score(y_test, pred)
print(f'f1 score : {f1:.3f}')
```

결과출력

```
f1 score : 0.780
```

F1 score 실습

- 앞서 작성한 get_clf_eval() 함수에 F1 스코어를 추가한다.
- 앞서 작성한 get_eval_by_threshold() 함수를 이용해 임계값 0.4~0.6 사이에서 평가 지표를 구한다.

```
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))
```

F1 score 실습 결과

- 앞서 작성한 코드의 출력을 표로 작성한다.

	0.4	0.45	0.5	0.55	0.6
Accuracy	0.8380	0.8492	0.8492	0.8659	0.8771
Precision	0.7286	0.7656	0.7742	0.8364	0.8824
Recall	0.8361	0.8033	0.7869	0.7541	0.7377
F1 score	0.7786	0.7840	0.7805	0.7931	0.8036

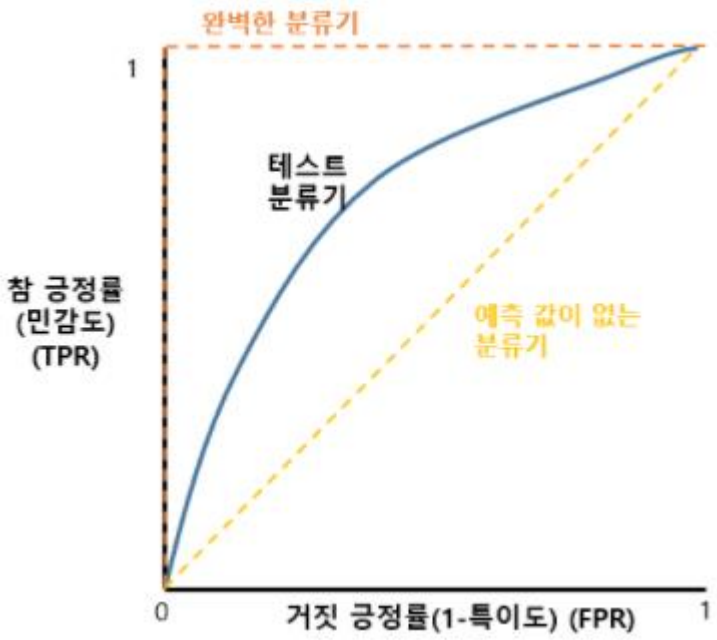
앞서 언급한 것처럼 F1 값은 precision과 recall의 조화 평균이므로 한쪽으로 치우쳐 지지 않을 때 높은 값을 보여준다. 따라서 임계값 0.6에서 가장 높은 F1 score 값을 보여주는 것을 확인할 수 있다.

ROC 곡선과 AUC

ROC(Receiver Operating Characteristic) curve 란?

- Roc곡선은 거짓 양성 비율에 대한 진짜 양성 비율의 곡선입니다.
- 주로 이진분류기 성능을 평가하는 지표로 사용됩니다.

ROC curve



X축-FPR(FalsePositiveRate) : 거짓 양성 비율

Y축-TPR(TruePositiveRate) : 진짜 양성 비율

! FPR = 1 - TNP = 1 - 특이성

ROC curve 축 해석

- FPR(FalsePositiveRate) : 실제 음성을 양성으로 잘못 예측 ➡ 생존을 사망이라고 판단

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

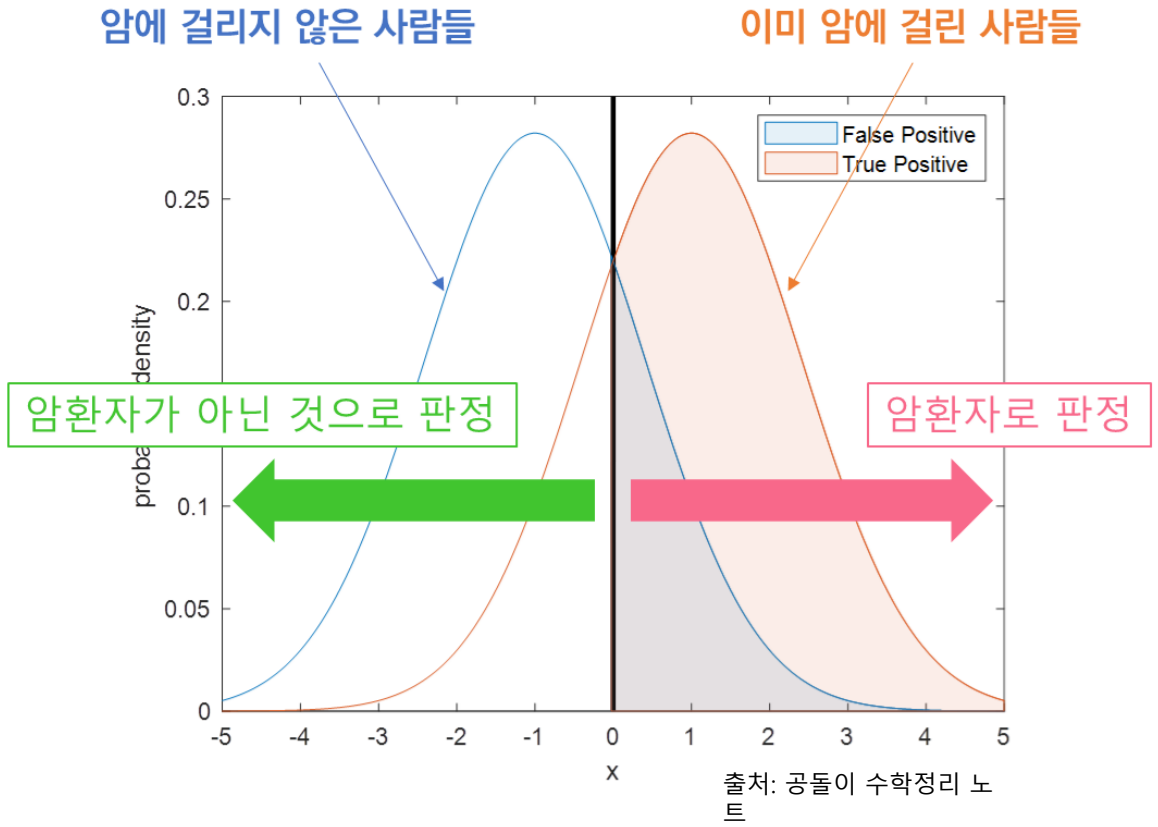
- TPR(TruePositiveRate) : 실제 양성을 제대로 양성이라고 예측 ➡ 생존을 생존이라고 판단

$$\frac{TP}{FN+TP}$$

- TNR(TrueNagativeRate) : 실제 음성을 제대로 음성이라고 예측 ➡ 사망을 사망이라고 판단

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

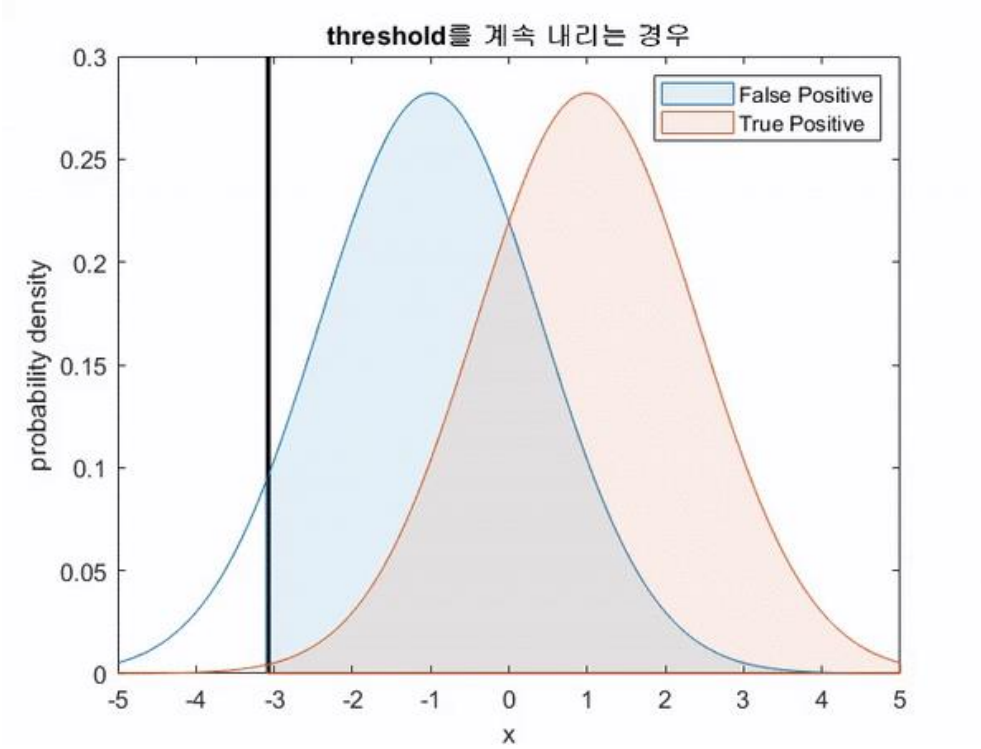
ROC curve TPR, FPR이해



출처: 공돌이 수학정리 노트

- False Positive와 True Positive의 가우시안 분포를 통하여 TPR과 FPR을 생각해 볼 수 있다.
- 임계점(threshold)에 따라서 TPR과 FPR이 변화한다.

ROC curve TPR과 FPR의 관계



출처: 공돌이 수학정리 노트

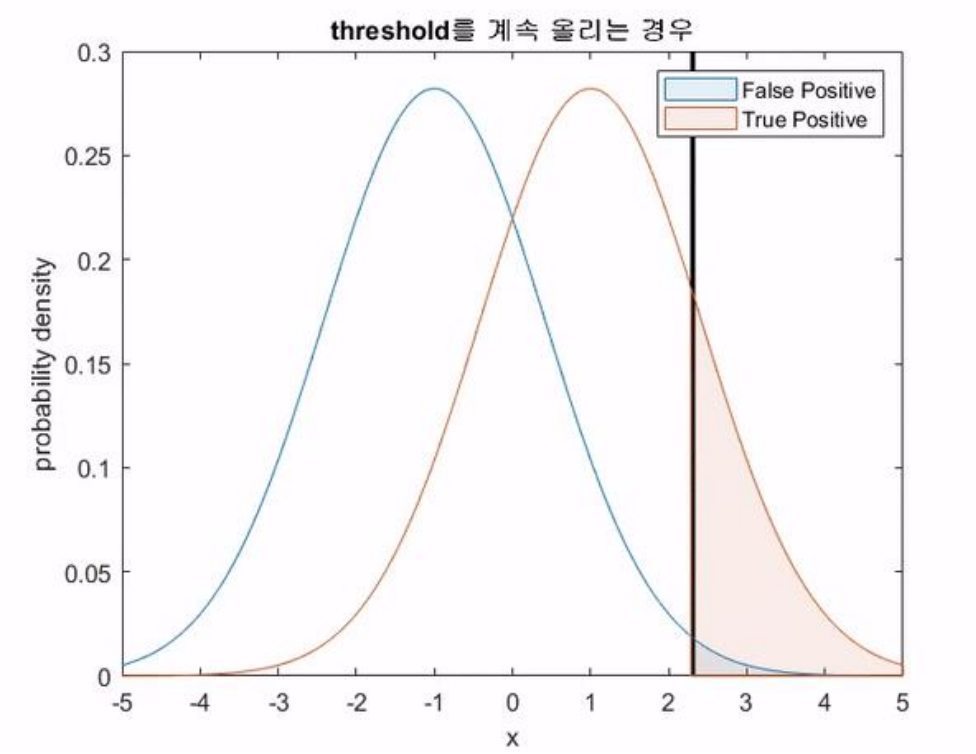
임계점(threshold)을 낮은 경우

1. 의사 A가 성격이 급해서 모든 환자들을 암환자라고 판단

2. 실제로 암이 걸린 환자 모두 암환자로 판단
   ➡ TPR값 상승

3. 암 걸리지 않은 환자도 모두 암환자로 판단
   ➡ FPR값 상승

ROC curve TPR과 FPR의 관계



출처: 공돌이 수학정리 노트

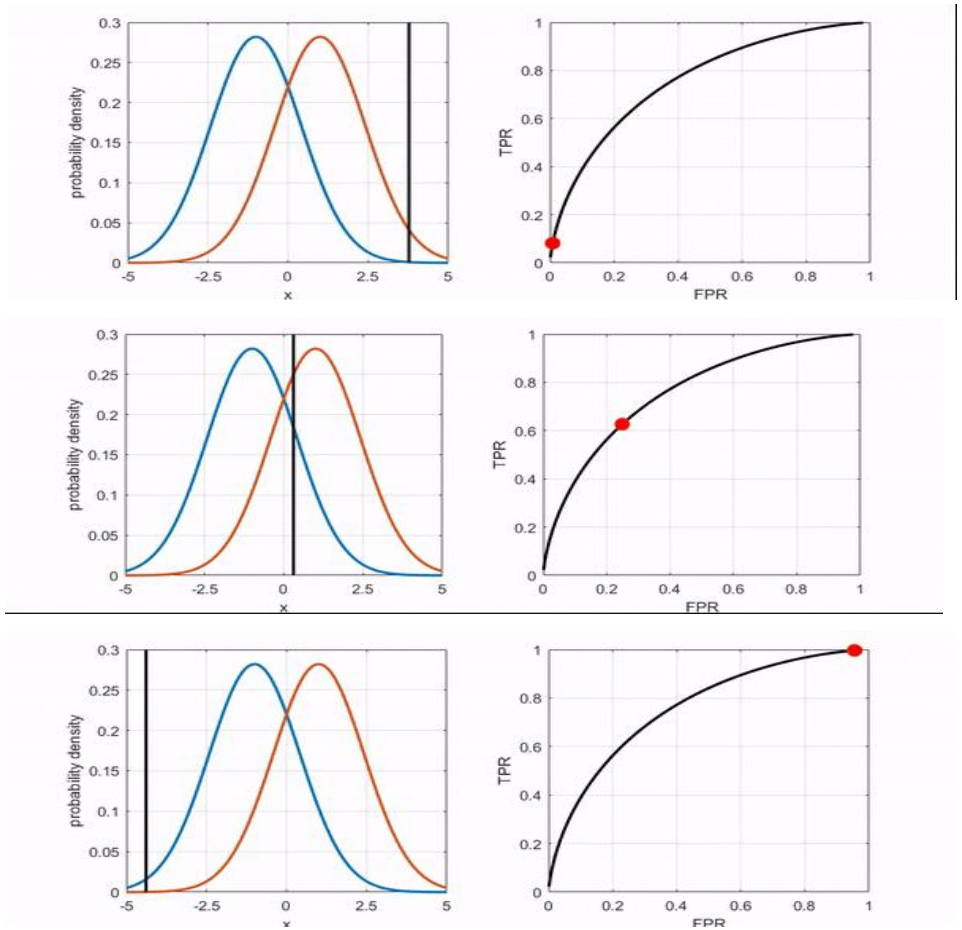
임계점(threshold)을 높은 경우

1. 의사 B가 성격이 소심해서 모든 환자들을 암이 아니라고 판단

2. 실제로 암이 걸린 환자 모두 정상으로 판단
   ➡ TPR값 감소

3. 암 걸리지 않은 환자도 모두 정상으로 판단
   ➡ FPR값 감소

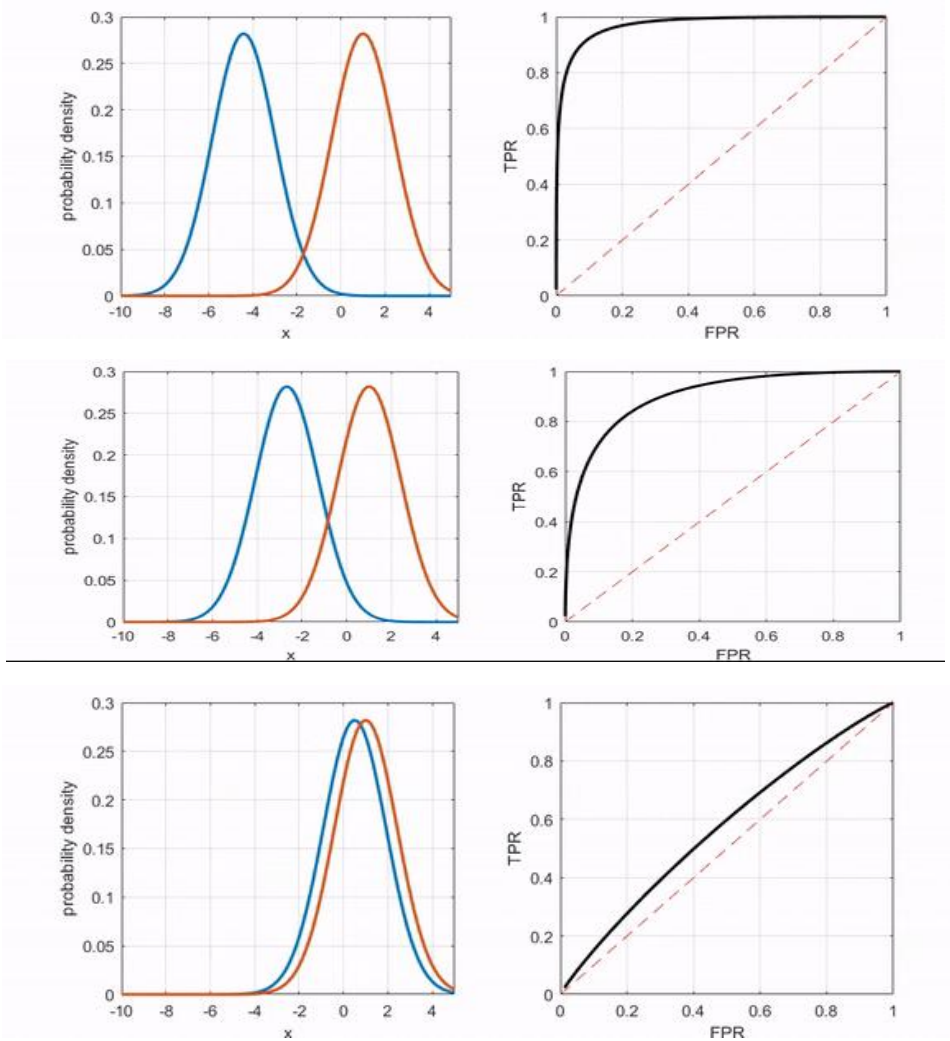
ROC curve 위의 점



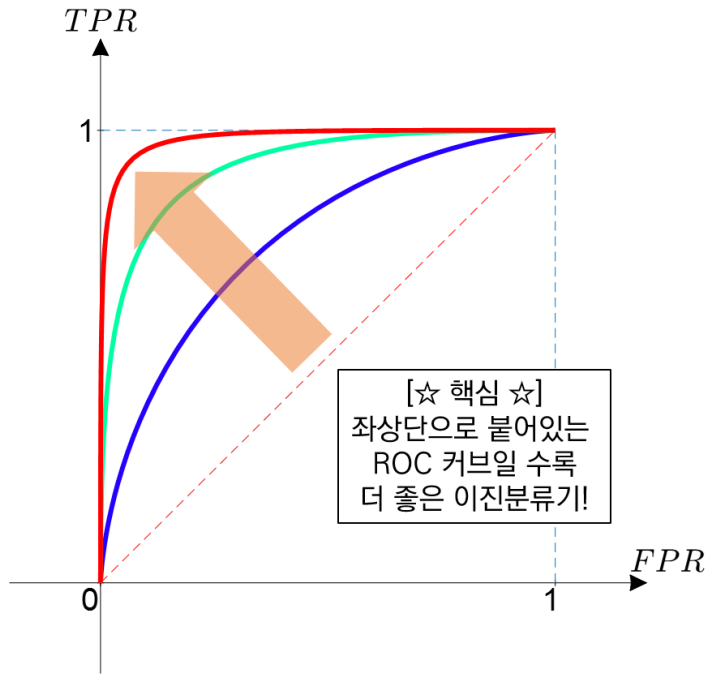
출처: 공돌이 수학정리 노트

- 앞서 봤듯이 임계값(threshold)이 변함에 따라서 FPR과 TPR의 값은 어느 정도 비례 관계적으로 함께 커지거나 작아진다는 것을 알 수 있다.
- threshold별 FPR과 TPR를 알아보는 것을 의미함
- 즉, 현재 이진 분류기의 분류성능은 변하지 않지만, 가능한 모든 임계값별 FPR과 TPR의 비율을 알 수 있다.

ROC curve 휨 정도



출처: 공돌이 수학정리 노트

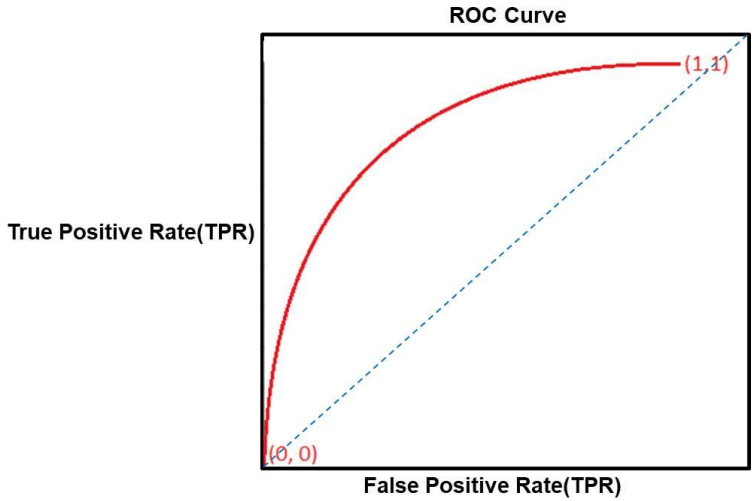


좌측 상단으로 갈 수록 좋은 분류기를 의미함!

ROC curve 그리기

임계값 0부터 1사이에서의 TPR과 FPR의 값을 통하여 ROC곡선을 그려준다.

임계값	0	모두 Positive 예측	TN=0	FPR=1



임계값	1	모두 Negative 예측	FP=0	FPR=0

ROC curve 그리기

사이킷런에서 roc_curve API를 제공한다.

```
from sklearn.metrics import roc_curve
fpr, tpr, thresholds = roc_curve(y_test, pred_proba_class1)
```

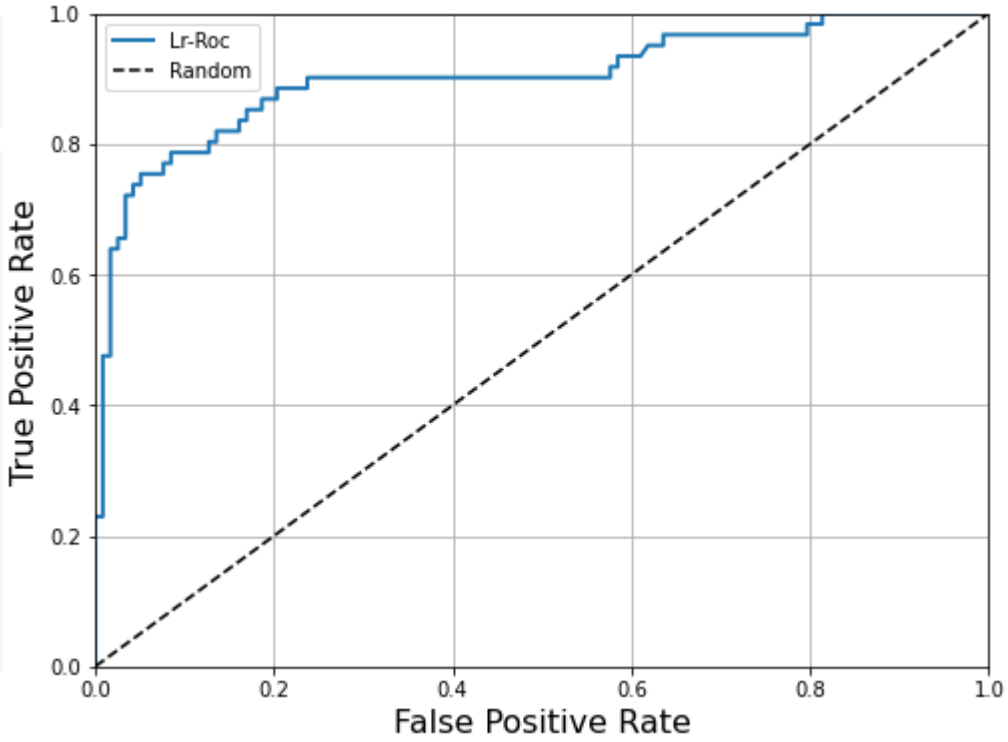
입력 파라미터	- y_true : 실제 클래스 값 array - y_score : predict_prob()의 반환 값 array
반환 값	- fpr : fpr 값을 array로 반환 - tpr : tpr 값을 array로 반환 - thresholds : thresholds 값을 array로 반환

ROC curve 실습

시각화를 위한 plot_roc_curve 함수를 정의하고 로지스틱 roc 곡선을 출력한다.

```
[ ]  from sklearn.metrics import roc_curve
     fpr, tpr, thresholds = roc_curve(y_test, pred_proba_class1)
```

```
[ ]  import matplotlib.pyplot as plt
     def plot_roc_curve(fpr, tpr, label=None):
         plt.plot(fpr, tpr, linewidth=2, label=label)
         plt.plot([0, 1], [0, 1], 'k--')
         plt.axis([0, 1, 0, 1])
         plt.xlabel('False Positive Rate', fontsize=16)
         plt.ylabel('True Positive Rate', fontsize=16)
         plt.grid(True)

     plt.figure(figsize=(8, 6))
     plot_roc_curve(fpr, tpr)
     plt.legend(['Lr-Roc', 'Random'], loc='best')
     plt.show()
```



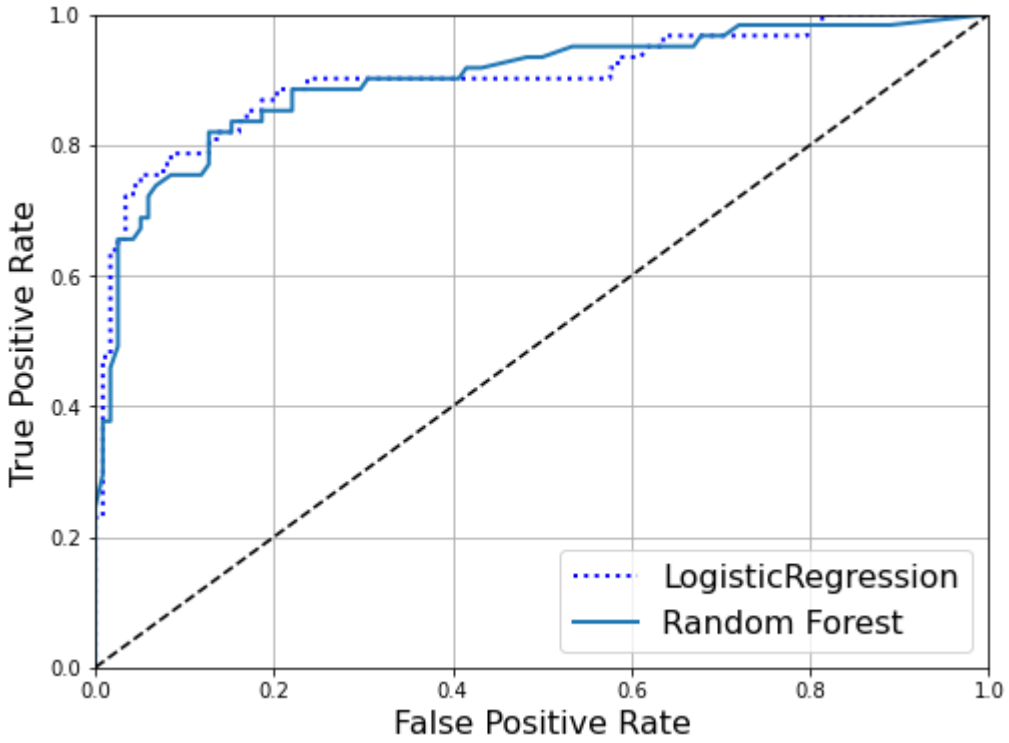
ROC curve 실습

랜덤 포레스트와 로지스틱의 roc곡선을 통한 성능 비교

```
from sklearn.ensemble import RandomForestClassifier

forest_clf = RandomForestClassifier(n_estimators=100, random_state=42)
forest_clf.fit(X_train, y_train)
y_probas_forest = forest_clf.predict_proba(X_test)
```

```
[ ] y_scores_forest = y_probas_forest[:, 1]
    fpr_forest, tpr_forest, thresholds_forest = roc_curve(y_test, y_scores_forest)
```

```
[ ] plt.figure(figsize=(8, 6))
    plt.plot(fpr, tpr, "b:", linewidth=2, label="LogisticRegression")
    plot_roc_curve(fpr_forest, tpr_forest, "Random Forest")
    plt.grid(True)
    plt.legend(loc="best", fontsize=16)
    plt.show()
```

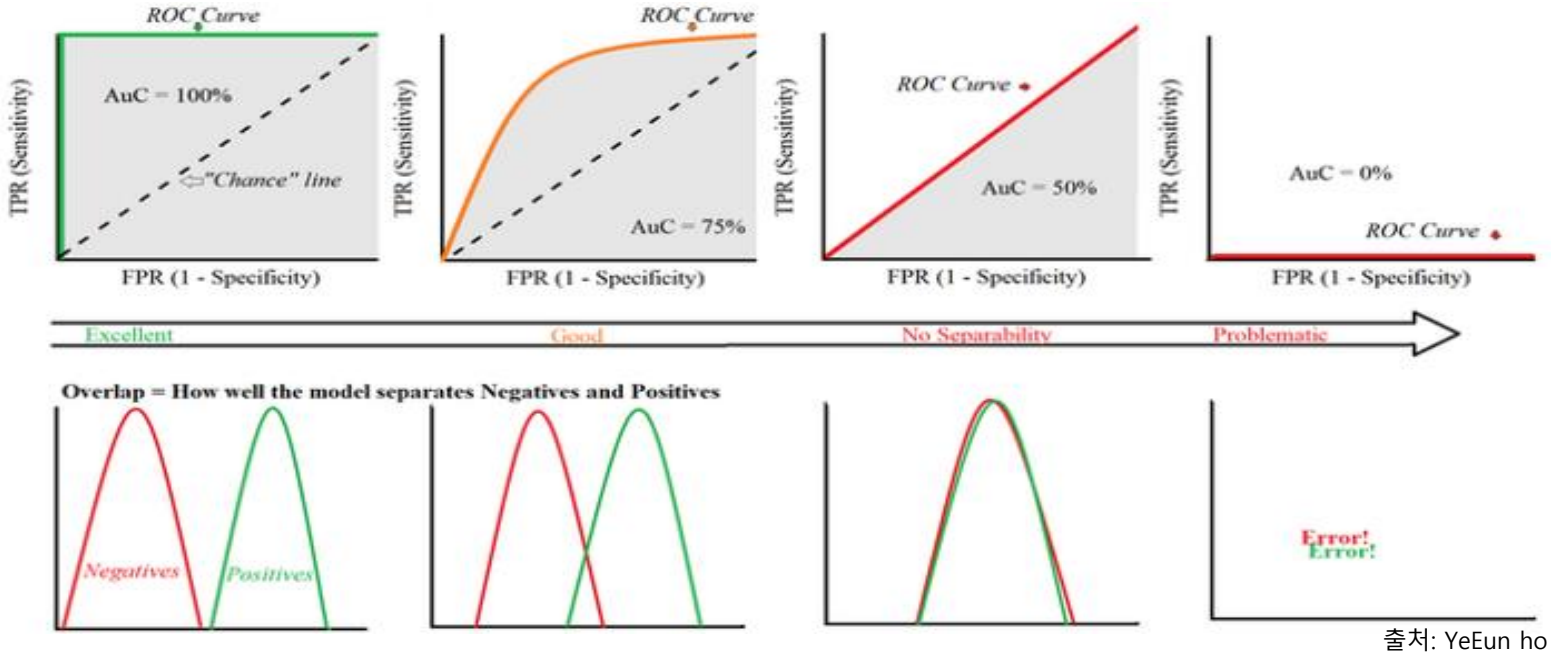


! 두 모델의 roc곡선이 겹치면서 어떤 것이 더 좋은 분류기 인지 ROC 곡선 만으로는 판단하기 어렵다!

AUC 값을 통하여 비교

AUC curve 실습

- AUC 값은 Roc곡선의 면적의 값이다.
- 앞서 언급했듯이 roc 커브는 임계값이 0 에서 1까지의 TPR과 FPR의 값의 변화를 그리는 곡선이다. 임계값이 0이면 모두 양성으로 예측하여 TPR과 FPR은 1이므로 (1,1) 지점에서 시작한다. 이때 auc는 roc의 아래 면적이므로 이 값이 커지려면 FPR 값이 작아질 때 TPR이 천천히 작아진다면 roc 커브는 (0,1) 지점에 가까울 것이고 이때 auc값이 가장 크면서 좋은 성능을 보이는 분류기라고 평가할 수 있다.



출처: YeEun ho

AUC curve 실습

앞서 본 랜덤 포레스트와 로지스틱의 AUC 값을 통한 비교

```
from sklearn.metrics import roc_auc_score

lr_pred_proba = lr_clf.predict_proba(X_test)[:,1]
fr_pred_proba = forest_clf.predict_proba(X_test)[:, 1]
lr_roc_score = roc_auc_score(y_test, lr_pred_proba)
fr_roc_score = roc_auc_score(y_test, fr_pred_proba)
print('lr_ROC AUC 값: {0:.4f}'.format(lr_roc_score))
print('fr_ROC AUC 값: {0:.4f}'.format(fr_roc_score))
```

```
lr_ROC AUC 값: 0.9024
fr_ROC AUC 값: 0.9001
```

	로지스틱	랜덤 포레스트
AUC	0.9024	0.9001

AUC값을 통하여 로지스틱 분류기가 더 좋은 성능을 보여준다고 판단!

3. 평가지표 활용
(피마 인디언 당뇨병 예측)

데이터 로드 및 타겟 변수 확인

```
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 = pd.read_csv('/content/diabetes.csv')
diabetes.head()
```

	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
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

타겟 변수

0 500 음성 클래스

1 268 양성 클래스

데이터 평가

로지스틱 모델을 사용하여 데이터를 평가한다.

```
[15] # 피처 데이터 셋 X, 레이블 데이터 셋 y 추출
     # 마지막 컬럼인 'Outcome' 컬럼은 레이블 값이므로 y로 설정
     X = diabetes.iloc[:,:-1]
     y = diabetes.iloc[:,-1]

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

     # LogisticRegression로 학습, 예측 진행
     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
```

타겟 변수가 음성 클래스가 더 많기 때문에 임계값을 조정하여 재현율을 높여준다.

데이터 평가

임계값을 조정하여 재현율을 확보한다.

```
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 )
```

기본 값

```
임곗값: 0.5
오차 행렬
[[88 12]
 [23 31]]
정확도: 0.7727, 정밀도: 0.7209, 재현율: 0.5741,    F1: 0.6392, AUC:0.7919
```

정확도와 정밀도는 높은 값을 보여주지만,
재현율이 너무 낮다!

변환 값

```
임곗값: 0.42
오차 행렬
[[82 18]
 [19 35]]
정확도: 0.7597, 정밀도: 0.6604, 재현율: 0.6481,    F1: 0.6542, AUC:0.7919
```

정확도와 정밀도는 어느 정도 떨어지지만
재현율이 대폭 상승하고 F1 값도 좋아진다!

감사합니다 ☺