모형 평가

4조

변영무, 이재욱, 조예빈, 한도담

목차

01 모형 평가

02 성능 평가 지표
- Confusion Matrix
  - 정확도
  - 정밀도
  - 재현율
- F1 score

03 ROC곡선과 AUC

1. 모형 평가

데이터 가공/변환

모델 학습/예측

평가

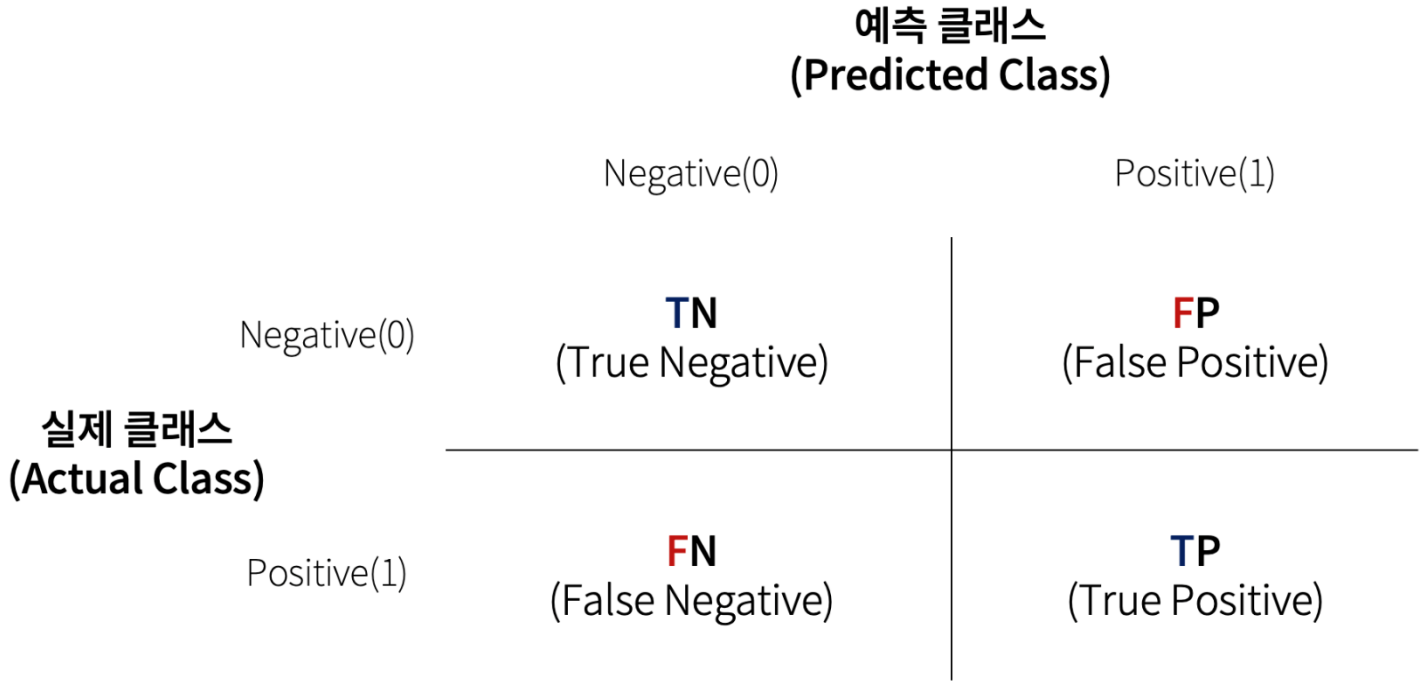
회귀 : Ch.5 – 오차평균합 등

분류 : 이진 분류의 성능 평가 지표
정확도, 오차행렬, 정밀도, 재현율 , F1 스코어, ROC AUC

2. 오차행렬(Confusion matrix, 혼동행렬)

오차행렬 : 타겟의 원래 클래스와 모형이 예측한 클래스가 일치하는지는 갯수로 센 결과를 표나 나타낸 것.

정답 클래스는 행(row)으로 예측한 클래스는 열(column)로 나타냄



(1) T :예측 '성공'

F : 예측 '실패'

(2) P : Positive라고 예측

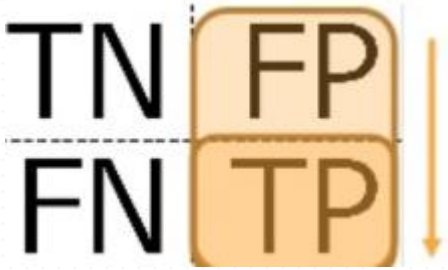
N : Negative라고 예측

- True Negative(TN): 실제 N인 정답을 N라고 예측 (정답) = 양성을 정확하게 분류
- False Positive(FP) : 실제 N인 정답을 P라고 예측 (오답) = 양성으로 잘못 분류
- False Negative(FN) : 실제 P인 정답을 N라고 예측 (오답) = 음성으로 잘못 분류
- True Positive(TP) : 실제 P인 정답을 P라고 예측 (정답) = 음성을 정확하게 분류

2. 오차행렬(Confusion matrix, 혼동행렬)

정확도 : 실제 데이터에서 예측 데이터가 얼마나 같은지

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



정밀도: Positive로 예측한 값들 중에 실제로 Positive한 값의 비율
FP ↓ 목적 ex) 스팸 메일 분류

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

재현율 : 실제 값이 Positive인 값들 중에 예측을 Positive로 한 값의 비율
FN ↓ 목적 ex) 암 진단

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

2. 오차행렬(Confusion matrix, 혼동행렬) 정확도 | 정밀도 | 재현율 | F1 score

```
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))
```

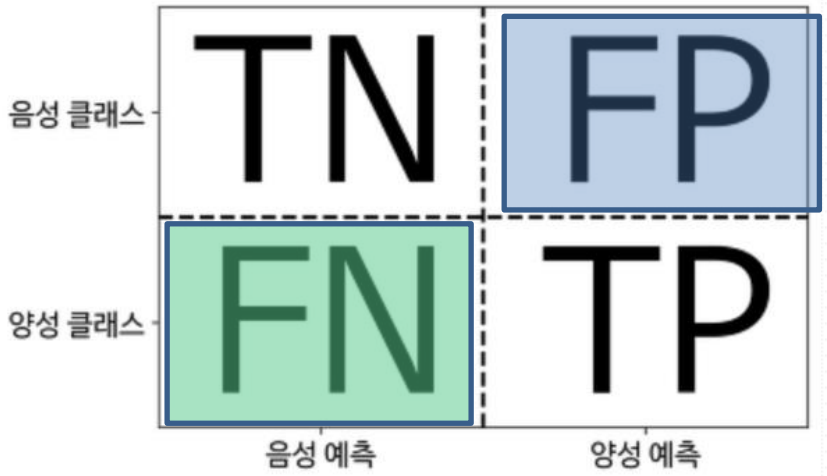
```
import numpy as np
import pandas as pd

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)
```

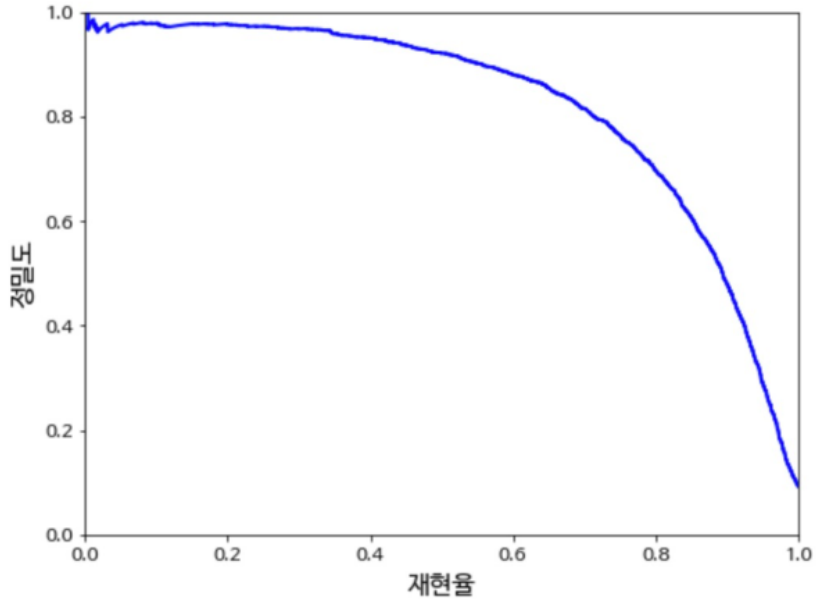
```
오차 행렬
[[108  10]
 [ 14  47]]
정확도: 0.8659, 정밀도: 0.8246, 재현율: 0.7705
```

- from sklearn.metrics import
  - confusion_matrix()
  - accuracy_score(y_test, _pred)
  - precision_score(y_test, _pred)
  - recall_score(y_test, pred)

2. 오차행렬(Confusion matrix, 혼동행렬) 정밀도와 재현율의 트레이드 오프

Precision(정밀도), Recall(재현율)의 Trade-Off : 적절한 수준에서 조정 필요





정밀도 ↑ 면 재현율 ↓

정밀도 ↓ 면 재현율 ↑

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

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

•정밀도 : 모델이 분류한 정답중에 진짜 정답이 얼만큼 있는지 측정

•재현율 : 실제 정답중에 모델이 정답을 얼만큼 분류했는지측정

2. 오차행렬(Confusion matrix, 혼동행렬)

정확도 만으로 모델 평가 가능?

	폐암 O	폐암 X	계
X-ray 양성	90	100	190
X-ray 음성	10	800	810
계	100	900	1000

유난히 낮은 Precision??

→ 데이터의 불균형 문제!

Accuracy(정확도) = (90 + 800) / 1000 = 0.89

VS

Precision(정밀도) = 90 / 190 = 0.47

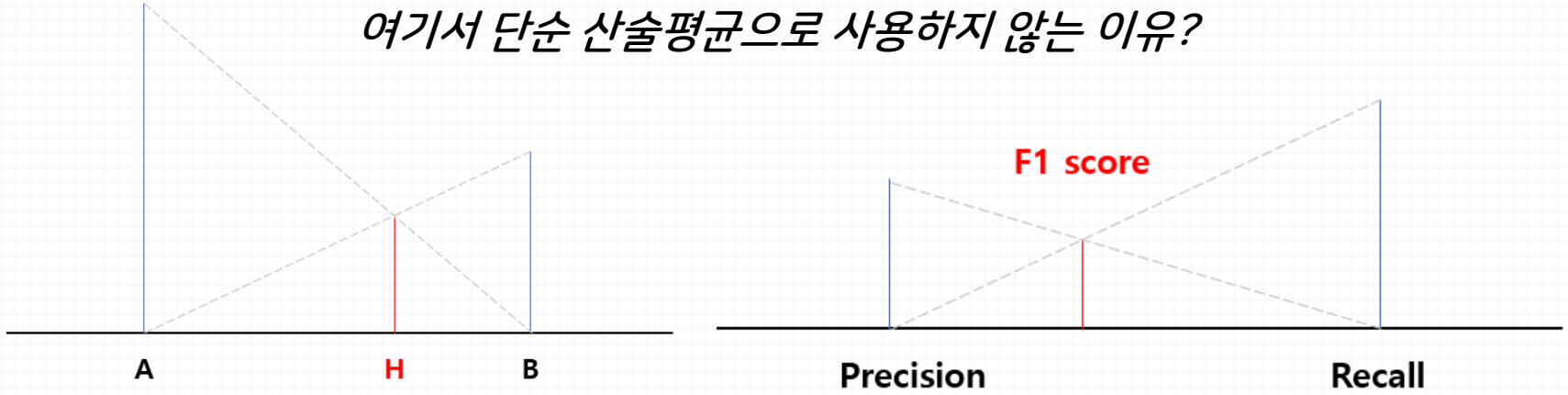
2. 오차행렬(Confusion matrix, 혼동행렬)

F1 score: Precision과 Recall의 조화평균

$$(F1\text{-}score) = 2 \times \frac{1}{\frac{1}{Precision} + \frac{1}{Recall}} = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

불균형 구조일 때, 모델의 성능을 정확하게 평가할 수 있으며, 성능을 하나의 숫자로 표현할 수 있음

여기서 단순 산술평균으로 사용하지 않는 이유?



서로 다른 길이의 A,B의 끝에서 다른 쪽의 base line으로 선을 내림,
이 때 만나는 점의 길이! 이 때, 작은 쪽보다도 낮은 평균이 나오게 됨.
결론적으로 조화평균은 산술평균을 이용하는 것보다 큰 비중이 끼치는 bias가 줄어들게 됨

2. 오차행렬(Confusion matrix, 혼동행렬) 정확도 | 정밀도 | 재현율 | F1 score

- from sklearn.metrics import
  - f1_score(y_test, pred)

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

```
F1 스코어: 0.7966
```

```
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
```

3. ROC 곡선과 AUC

ROC(Receiver Operating Characteristic 곡선:

- x축 FPR(False Positive Rate), Y축 TPR(True Positive Rate)
- FPR이 변할때 TPR이 어떻게 변하는지를 나타내는 곡선
- 45도 직선에 가까울수록 좋지 않은 성능을 가짐

특이성 : 실제값 Negative(음성)가 정확히 예측 되어야하는 수준

$$TNR\,(\text{specificity}) = \frac{TN}{FP + TN}$$

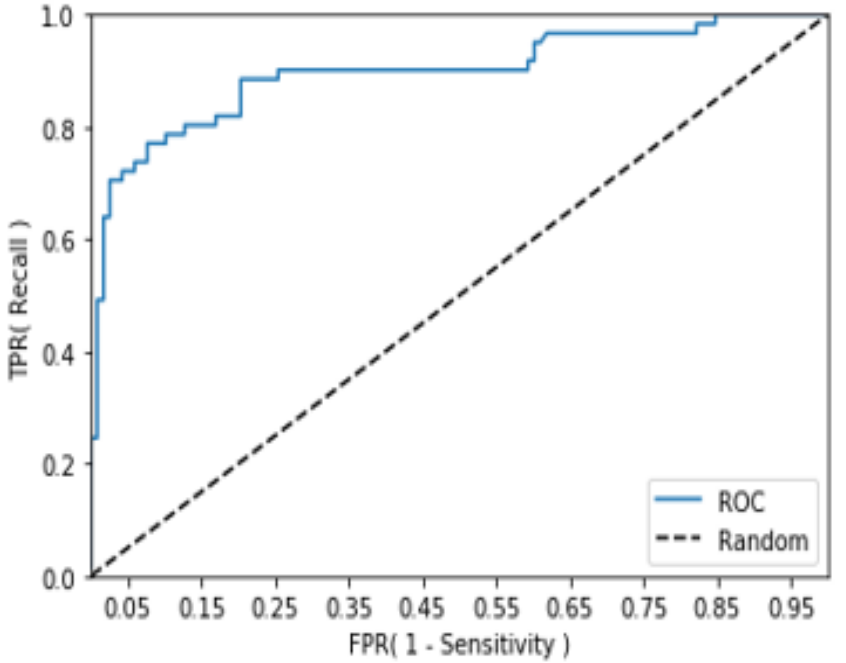
1-특이성 :

$$\text{FPR (1-specificity)} = \frac{\text{FP}}{\text{TN} + \text{FP}}$$

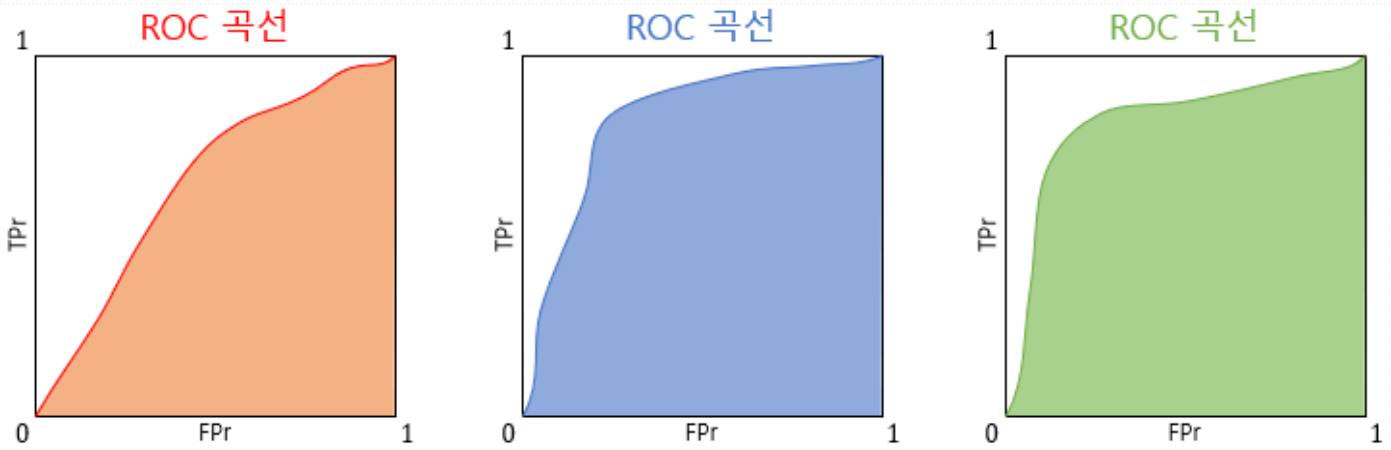
민감도 : 실제값 Positive(양성)가 정확히 예측되어야 하는 수준

(= 재현율)

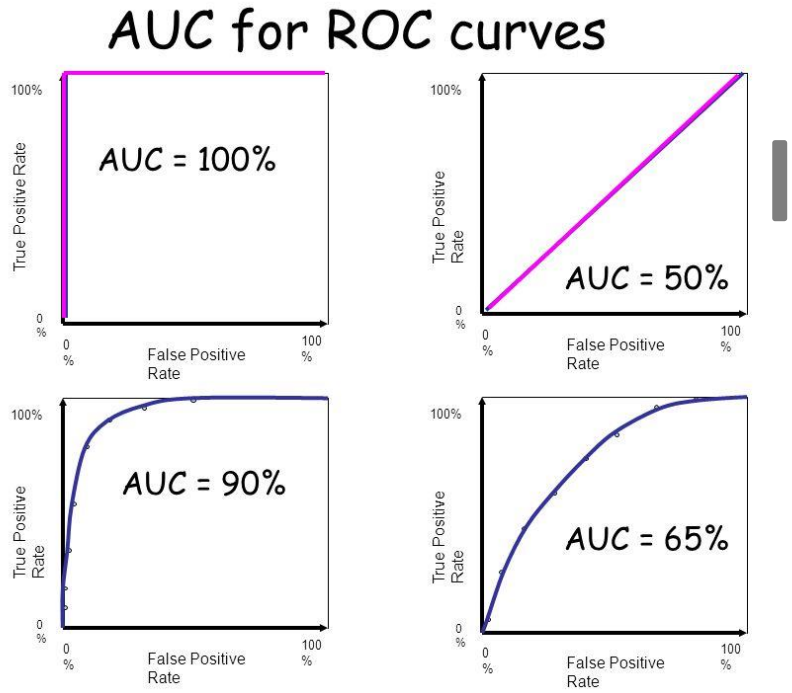
$$\text{TPR (sensitivity)} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$



3. ROC 곡선과 AUC



적색은 청색이나 녹색보다
안 좋다는 것을 쉽게 알 수 있음
But. 청색과 녹색 중 하나를 선택?



AUC(Area Under Curve) : ROC 곡선하의 면적

0.90 ~ 1.00 = Excellent
0.80 ~ 0.90 = Good
0.70 ~ 0.80 = Fair
0.60 ~ 0.70 = Poor
0.50 ~ 0.60 = Fail

ROC 그래프의 **밑부분 면적이 넓을수록 GOOD**

(x값은 작을수록, y값은 클수록 good이니까!)

3. ROC 곡선과 AUC

ROC 곡선

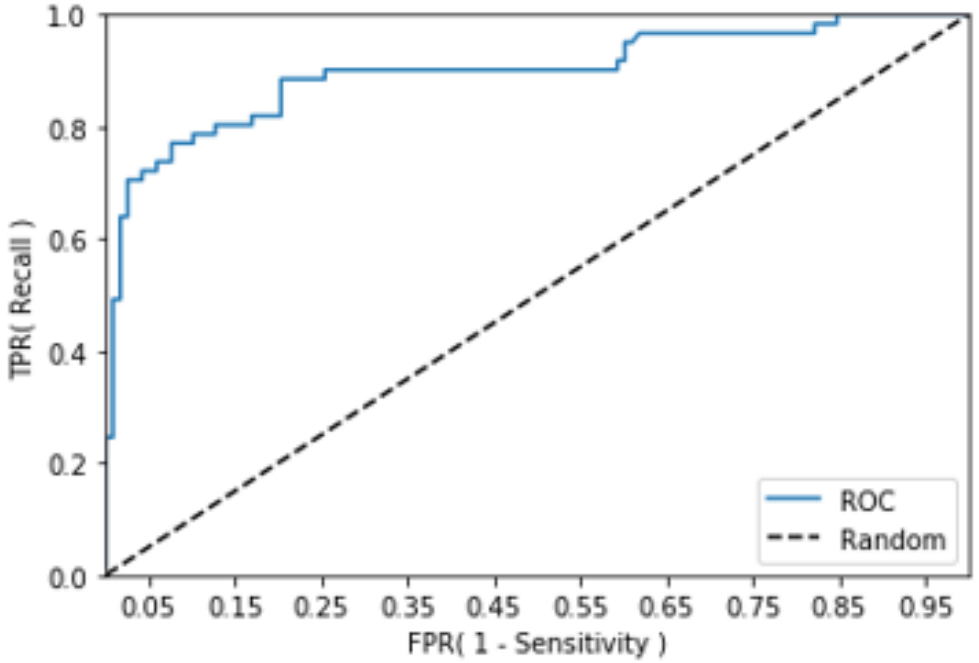
```
def roc_curve_plot(y_test , pred_proba_c1):
    # 임곗값에 따른 FPR, TPR 값을 반환 받음.
    fprs , tprs , thresholds = roc_curve(y_test ,pred_proba_c1)

    # 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] )
```

AUC

```
from sklearn.metrics import roc_auc_score

pred = lr_clf.predict(X_test)
roc_score = roc_auc_score(y_test, pred)
print('ROC AUC 값: {0:.4f}'.format(roc_score))
```

```
ROC AUC 값: 0.8429
```

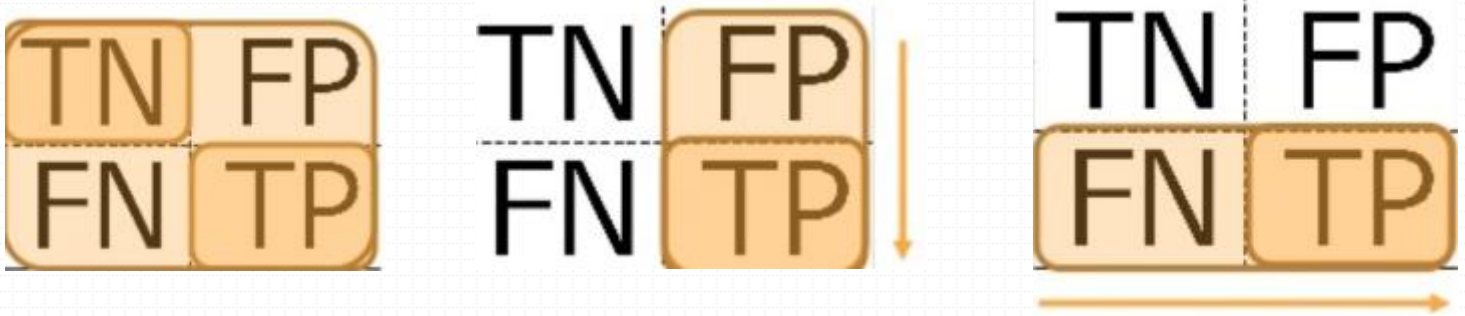
- roc_auc_score(y_test, pred)



요약

성능 평가 지표

- Confusion Matrix



Accuracy 정확도 | Precision 정밀도 | Recall 재현율

- confusion_matrix()
- accuracy_score(y_test, _pred)
- precision_score(y_test, _pred)
- recall_score(y_test, pred)
- f1_score(y_test, pred)
- roc_auc_score(y_test, pred)

- F1 score : 불균형 데이터일때 사용, 정밀도와 재현율의 조화평균

$$(F1\text{-}score) = 2\times\frac{1}{\frac{1}{Precision}+\frac{1}{Recall}} = 2\times\frac{Precision \times Recall}{Precision + Recall}$$

- ROC곡선 :x축 1-특이도, y축 민감도

- AUC : ROC 곡선 아래 전체 영역을 측정한 값