

3강: 모델 성능 평가의 이해

인공지능 일반강좌 : 기계학습의 이해(L2-1)

Contents

혼동행렬의 이해

혼동행렬 실습

타이타닉 생존자 예측

MNIST 데이터 적용

혼동 행렬 연습 II

파마인디언 당뇨병 예측

Confusion Matrix 이해

- 임신테스트 예제로 임신이면 +(1), 아니면 -(0)
 - ✓ TP (True Positive) : 임신 했을 때, 테스트 결과는 1
 - ✓ TN (True Negative) : 임신은 안 했을 때 , 테스트 결과는 0
 - ✓ FP (False Positive) : 임신은 안 했을 때, 테스트 결과는 1 (에러)
 - ✓ FN (False Negative) : 임신 했는데, 테스트 결과는 0



성능 평가 (1)

- 분류문제는 회귀 분석과 달리 다양한 성능평가 기준이 필요하다.
 - ✓ 이 절에서는 분류문제에 사용되는 다양한 성능평가 기준에 대해 알아본다.
- 사이킷런 패키지에서 지원하는 분류 성능평가 명령
 - ✓ `confusion_matrix(y_true, y_pred)`
 - ✓ `accuracy_score(y_true, y_pred)`
 - ✓ `precision_score(y_true, y_pred)`
 - ✓ `recall_score(y_true, y_pred)`
 - ✓ `fbeta_score(y_true, y_pred, beta)`
 - ✓ `f1_score(y_true, y_pred)`
 - ✓ `classification_report(y_true, y_pred)`
 - ✓ `roc_curve`
 - ✓ `auc`

혼동행렬 Confusion Matrix

- 혼동행렬

- ✓ 타겟의 원래 클래스와 모형이 예측한 클래스가 일치하는지는 개수로 센 결과를 행렬로 나타낸 것
- ✓ 정답 클래스는 행(row)으로 예측한 클래스는 열(column)로 나타낸다.

	예측 클래스	예측 클래스 1	예측 클래스 2
정답 클래스 0	정답 클래스가 0, 예측 클래스가 0인 표본의 수	정답 클래스가 0, 예측 클래스가 1인 표본의 수	정답 클래스가 0, 예측 클래스가 2인 표본의 수
정답 클래스 1	정답 클래스가 1, 예측 클래스가 0인 표본의 수	정답 클래스가 1, 예측 클래스가 1인 표본의 수	정답 클래스가 1, 예측 클래스가 2인 표본의 수
정답 클래스 2	정답 클래스가 2, 예측 클래스가 0인 표본의 수	정답 클래스가 2, 예측 클래스가 1인 표본의 수	정답 클래스가 2, 예측 클래스가 2인 표본의 수

혼동행렬 Confusion Matrix (1)

```
In [13]: from sklearn.metrics import confusion_matrix
```

```
y_true = [2, 0, 2, 2, 0, 1]  
y_pred = [0, 0, 2, 2, 0, 2]
```

예측은 2이지만, 실제는 1임

```
In [14]: confusion_matrix(y_true, y_pred)
```

```
Out[14]: array([[2, 0, 0],  
               [0, 0, 1],  
               [1, 0, 2]], dtype=int64)
```

예측은 0, 실제는 2

	예측 0	예측 1	예측 2
실제 0	2	0	0
실제 1	0	0	1
실제 2	1	0	2

혼동행렬 Confusion Matrix (2)

제품을 생산하는 공장에서는 완성된 제품에 대해 품질 테스트를 실시

- 불량품을 찾아내고 찾아낸 불량품은 공장으로 되돌린다(리콜, recall)
- 이 때 품질 테스트 결과가 양성이면 불량품이라고 예측한 것이고
- 음성이고 정상제품이라고 예측한 것
 - True Positive: 불량품을 불량품이라고 정확하게 예측
 - True Negative: 정상제품을 정상제품이라고 정확하게 예측
 - False Positive: 불량품을 정상제품이라고 잘못 예측
 - False Negative: 정상제품을 불량품이라고 잘못 예측

	불량품이라고 예측	정상제품이라고 예측
실제로 불량품	True Positive	False Negative
실제로 정상제품	False Positive	True Negative

	양성이라고 예측	음성이라고 예측
실제 양성	양성 예측이 맞음 (True Positive)	음성 예측이 틀림 (False Negative)
실제 음성	양성 예측이 틀림 (False Positive)	음성 예측이 맞음 (True Negative)

혼동행렬 Confusion Matrix (3)

- 예제) 암(cancer, 악성종양)을 검진
 - ✓ 암에 걸린 것을 양성(陽性, positive)이라 함
 - ✓ 걸리지 않은 것을 음성이라고 함.
 - 종양(tumor)의 양성(良性, benign), 악성(惡性, malignant) 용어와 다르다는 점에 주의하라.
 - ✓ True Positive: 암을 암이라고 정확하게 예측
 - ✓ True Negative: 암이 아닌것을 암이 아니라고 정확하게 예측
 - ✓ False Positive: 암을 암이 아니라고 잘못 예측
 - ✓ False Negative: 암이 아닌것을 암이라고 잘못 예측

	암이라고 예측	암이 아니라고 예측
실제로 암	True Positive	False Negative
실제로 암이 아님	False Positive	True Negative

혼동행렬 Confusion Matrix (4)

- FDS(Fraud Detection System)
 - ✓ 금융 거래, 회계 장부 등에서 잘못된 거래, 사기거래를 찾아내는 시스템
 - ✓ FDS의 예측 결과가 양성이면 사기거래라고 예측
 - ✓ 음성이면 정상거래라고 예측
 - True Positive: 사기를 사기라고 정확하게 예측
 - True Negative: 정상을 정상이라고 정확하게 예측
 - False Positive: 정상을 사기라고 잘못 예측
 - False Negative: 사기를 정상이라고 잘못 예측

	사기라고 예측	정상이라고 예측
실제로 사기	True Positive	False Negative
실제로 정상	False Positive	True Negative

혼동행렬 Confusion Matrix (5)

```
from sklearn.metrics import classification_report
```

```
y_true = [0, 0, 0, 1, 1, 0, 0]  
y_pred = [0, 0, 0, 0, 1, 1, 1]
```

실제 0인 데이터 중의 60%만 0으로 판별되었고,
실제 1인 데이터 중의 50%만 1로 판별되었음을 알 수 있다

```
print(classification_report(y_true, y_pred, target_names=['class 0', 'class 1']))
```

	precision	recall	f1-score	support
class 0	0.75	0.60	0.67	5
class 1	0.33	0.50	0.40	2

- macro: 단순평균
- weighted: 각 클래스에 속하는 표본의 갯수로 가중평균
- accuracy: 정확도. 전체 학습데이터의 개수에서 각 클래스에서 자신의 클래스를 정확하게 맞춘 개수의 비율

accuracy			0.57	7
macro avg	0.54	0.55	0.53	7
weighted avg	0.63	0.57	0.59	7

결과에서 0이라고 예측한 데이터의 75%만 실제로 0이었고 1이라고 예측한 데이터의 33%만 실제로 1이다.

혼동행렬 Confusion Matrix (6)

	예측 클래스 0	예측 클래스 1	예측 클래스 2
정답 클래스 0	TP	FP/FN	FP/FN
정답 클래스 1	FP/FN	TP	FP/FN
정답 클래스 2	FP/FN	FP/FN	TP

```

y_true = [0, 0, 1, 1, 2, 2, 2]
y_pred = [0, 0, 1, 2, 2, 2, 1]
target_names = ['class 0', 'class 1', 'class 2']
print(classification_report(y_true, y_pred, target_names=target_names))

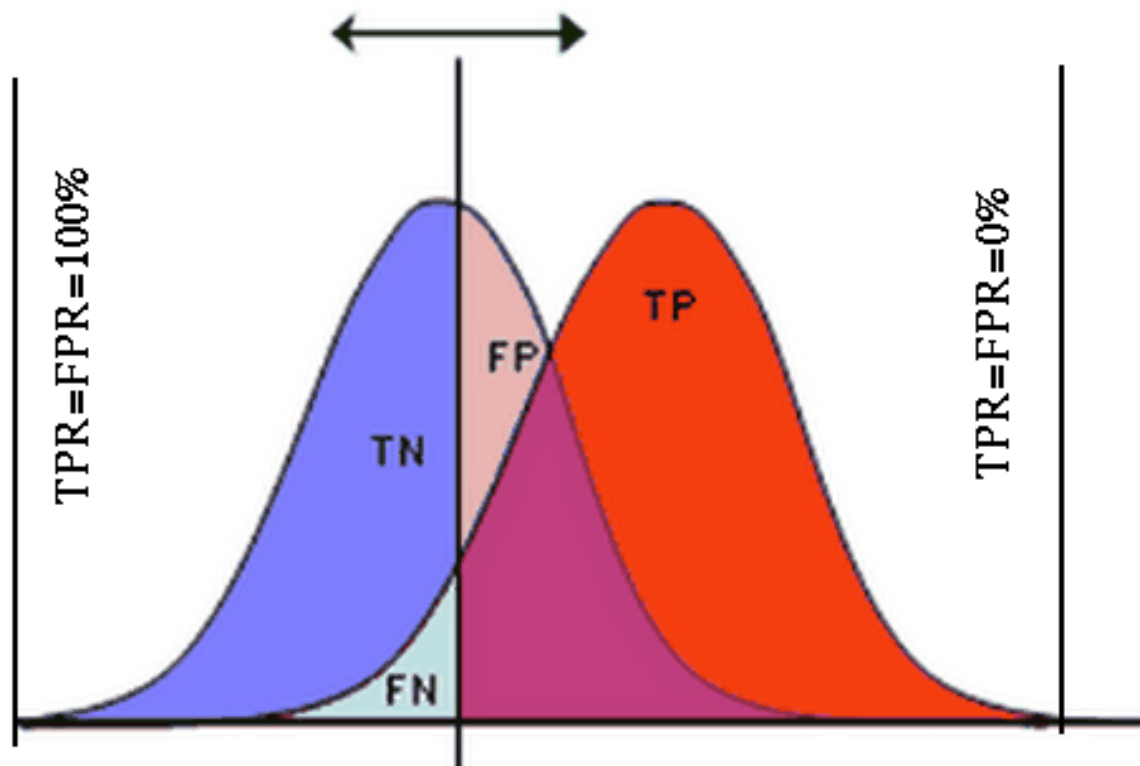
```

	precision	recall	f1-score	support
class 0	1.00	1.00	1.00	2
class 1	0.50	0.50	0.50	2
class 2	0.67	0.67	0.67	3
accuracy			0.71	7
macro avg	0.72	0.72	0.72	7
weighted avg	0.71	0.71	0.71	7

ROC 커브 (1)

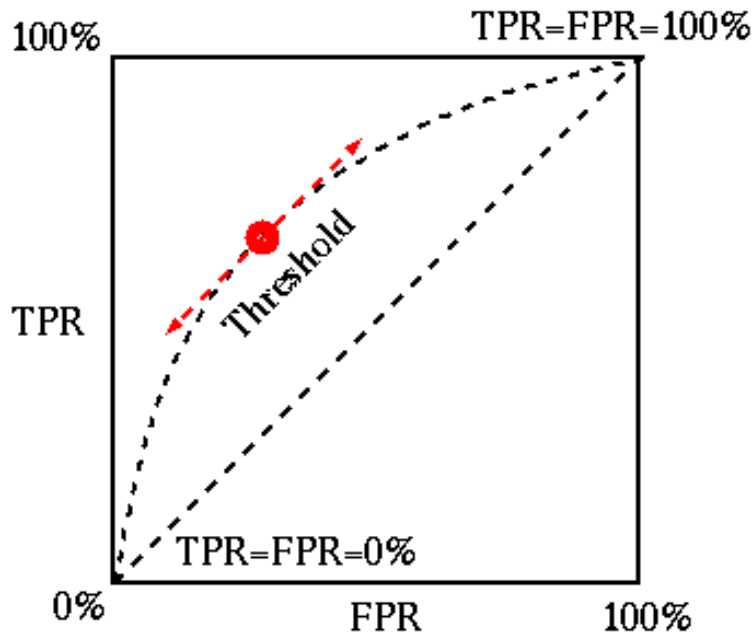
- ROC(Receiver Operator Characteristic) 커브는
 - ✓ 재현율(recall)과 위양성률(fall-out)은 일반적으로 양의 상관 관계가 있다.
 - ✓ 재현율을 높이기 위해서는 양성으로 판단하는 기준(threshold)을 낮추어 약간의 증거만 있어도 양성으로 판단하도록 하면 된다.
 - ✓ 그러나 이렇게 되면 음성임에도 양성으로 판단되는 표본 데이터가 같이 증가하게 되어 위양성율이 동시에 증가한다.
 - ✓ 반대로 위양성율을 낮추기 위해 양성을 판단하는 기준을 엄격하게 두게 되면 증거 부족으로 음성 판단을 받는 표본 데이터의 수가 같이 증가하므로 재현율이 떨어짐

ROC 커브 (2)



ROC 커브 (3)

- 판별함수 (discriminant function)
 - ✓ 모든 이진 분류 모형은 판별 평면으로부터의 거리에 해당하는 판별함수를 가지며
 - ✓ 판별함수값이 음수이면 0인 클래스, 양수이면 1인 클래스에 해당한다고 판별
 - ROC 커브는 이 클래스 판별 기준 값이 달라진다면 결과가 어떻게 달라짐을 표현



ROC 커브 (4)

```
import pandas as pd
import numpy as np
import matplotlib as plt
```

```
from sklearn.datasets import make_classification
from sklearn.linear_model import LogisticRegression
```

```
X, y = make_classification(n_samples=16, n_features=2,
                          n_informative=2, n_redundant=0,
                          random_state=0)
```

```
model = LogisticRegression().fit(X, y)
y_hat = model.predict(X)
f_value = model.decision_function(X)
```

ROC 커브 (5)

```
df = pd.DataFrame(np.vstack([f_value, y_hat, y]).T, columns=["f", "y_hat", "y"])
df.sort_values("f", ascending=False).reset_index(drop=True)
```

	f	y_hat	y				
0	2.363163	1.0	1.0	9	-0.765888	0.0	0.0
1	2.065047	1.0	1.0	10	-0.926932	0.0	1.0
2	1.633657	1.0	1.0	11	-1.046770	0.0	0.0
3	1.626171	1.0	1.0	12	-1.214700	0.0	0.0
4	1.624967	1.0	1.0	13	-1.429382	0.0	0.0
5	1.219678	1.0	1.0	14	-2.081586	0.0	0.0
6	0.378296	1.0	0.0	15	-4.118969	0.0	0.0
7	0.094285	1.0	1.0				
8	-0.438666	0.0	0.0				

ROC 커브 (6)

- ROC 커브는 현재는 0을 기준값(threshold)으로 클래스를 구분하여, 값이 0보다 크면 양성, 작으면 음성
- 데이터 분류가 다르게 되도록 기준값을 증가 혹은 감소시킨다. 위의 표에서는 기준값을 0.244729보다 크도록 올리면 6번 데이터는 더이상 양성이 아니다.
- 기준값을 여러가지 방법으로 증가 혹은 감소시키면서 이를 반복하면 여러가지 다른 기준값에 대해 분류 결과가 달라지고 재현율, 위양성률 등의 성능평가 점수도 달라진다.
- 기준값 0을 사용하여 이진 분류결과표, 재현율, 위양성율을 계산하면 다음과 같다.

```
► confusion_matrix(y, y_hat, labels=[1, 0])
```

```
] array([[7, 1],  
        [1, 7]], dtype=int64)
```

```
► recall = 6 / (6 + 2)  
fallout = 1 / (1 + 7)  
print("recall =", recall)  
print("fallout =", fallout)
```

```
recall = 0.75  
fallout = 0.125
```

ROC 커브 (7)

```
from sklearn.metrics import roc_curve
```

```
fpr, tpr, thresholds = roc_curve(y, model.decision_function(X))  
fpr, tpr, thresholds
```

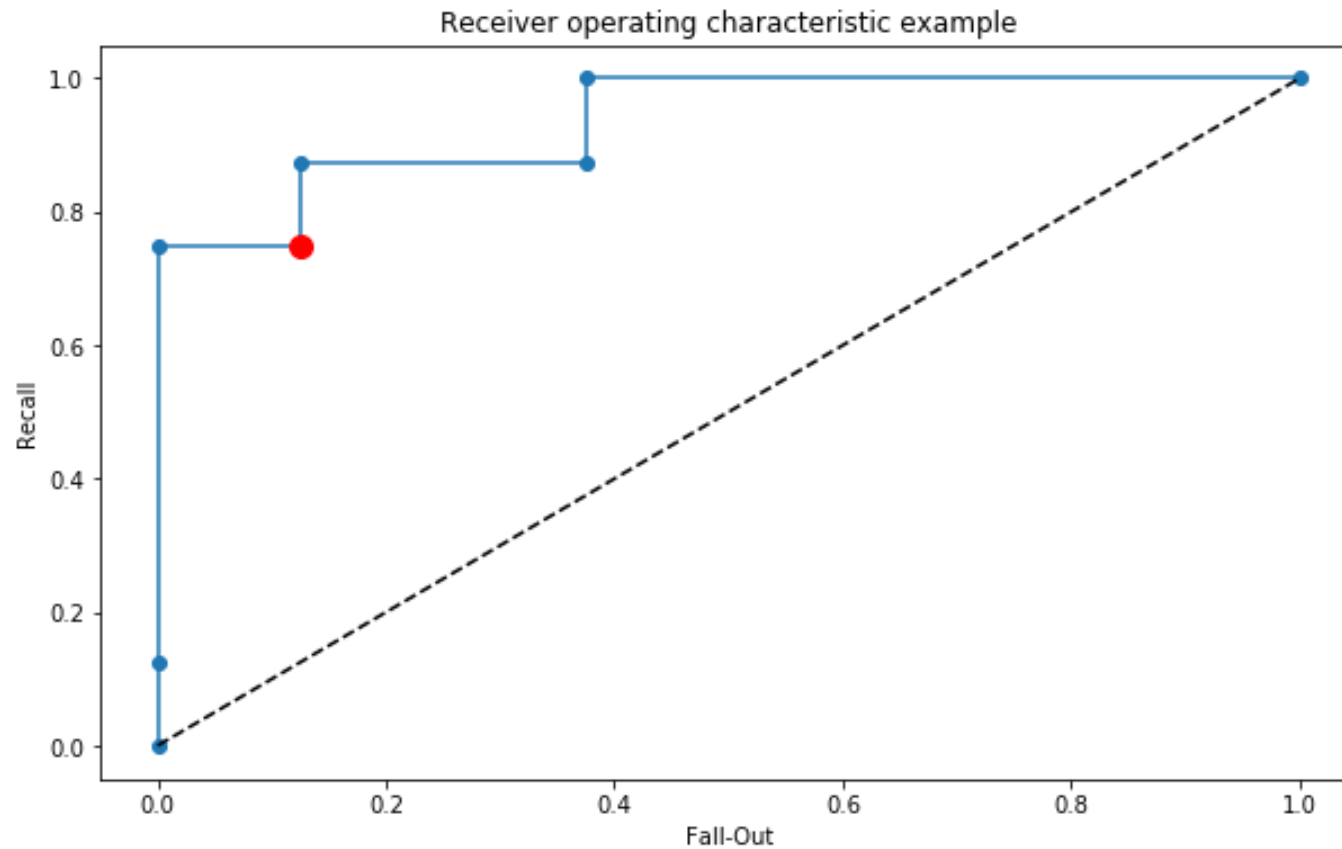
```
(array([0.    , 0.    , 0.    , 0.125, 0.125, 0.375, 0.375, 1.    ]),  
 array([0.    , 0.125, 0.75 , 0.75 , 0.875, 0.875, 1.    , 1.    ]),  
 array([ 3.36316277,  2.36316277,  1.21967832,  0.37829565,  0.09428499,  
        -0.76588836, -0.92693183, -4.11896895]))
```

decision_function 메서드를 제공하지 않는 모형은 predict_proba 명령을 써서 확률을 입력

```
fpr, tpr, thresholds = roc_curve(y, model.predict_proba(X)[:, 1])  
fpr, tpr, thresholds
```

```
(array([0.    , 0.    , 0.    , 0.125, 0.125, 0.375, 0.375, 1.    ]),  
 array([0.    , 0.125, 0.75 , 0.75 , 0.875, 0.875, 1.    , 1.    ]),  
 array([1.9139748 , 0.9139748 , 0.77200693, 0.59346197, 0.5235538 ,  
        0.31736921, 0.28354759, 0.01600107]))
```

ROC 커브 (8)



ROC 커브 (9)

정확도, 정밀도, 재현도 등의 성능이 동일한 모형도, ROC 커브에서 살펴보면 성능이 다름

```
from sklearn.datasets import make_classification
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC

X, y = make_classification(n_samples=1000,
                           weights=[0.95, 0.05], random_state=5)

model1 = LogisticRegression().fit(X, y)
y_hat1 = model1.predict(X)

model2 = SVC(gamma=0.0001, C=3000, probability=True).fit(X, y)
y_hat2 = model2.predict(X)
```

```
print(confusion_matrix(y, y_hat1))
```

```
[[940  3]
 [ 30 27]]
```

```
print(confusion_matrix(y, y_hat2))
```

```
[[940  3]
 [ 30 27]]
```

ROC 커브 (10)

```
print(classification_report(y, model1.predict(X)))
```

	precision	recall	f1-score	support
0	0.97	1.00	0.98	943
1	0.90	0.47	0.62	57
accuracy			0.97	1000
macro avg	0.93	0.74	0.80	1000
weighted avg	0.97	0.97	0.96	1000

```
print(classification_report(y, model2.predict(X)))
```

	precision	recall	f1-score	support
0	0.97	1.00	0.98	943
1	0.90	0.47	0.62	57
accuracy			0.97	1000
macro avg	0.93	0.74	0.80	1000
weighted avg	0.97	0.97	0.96	1000

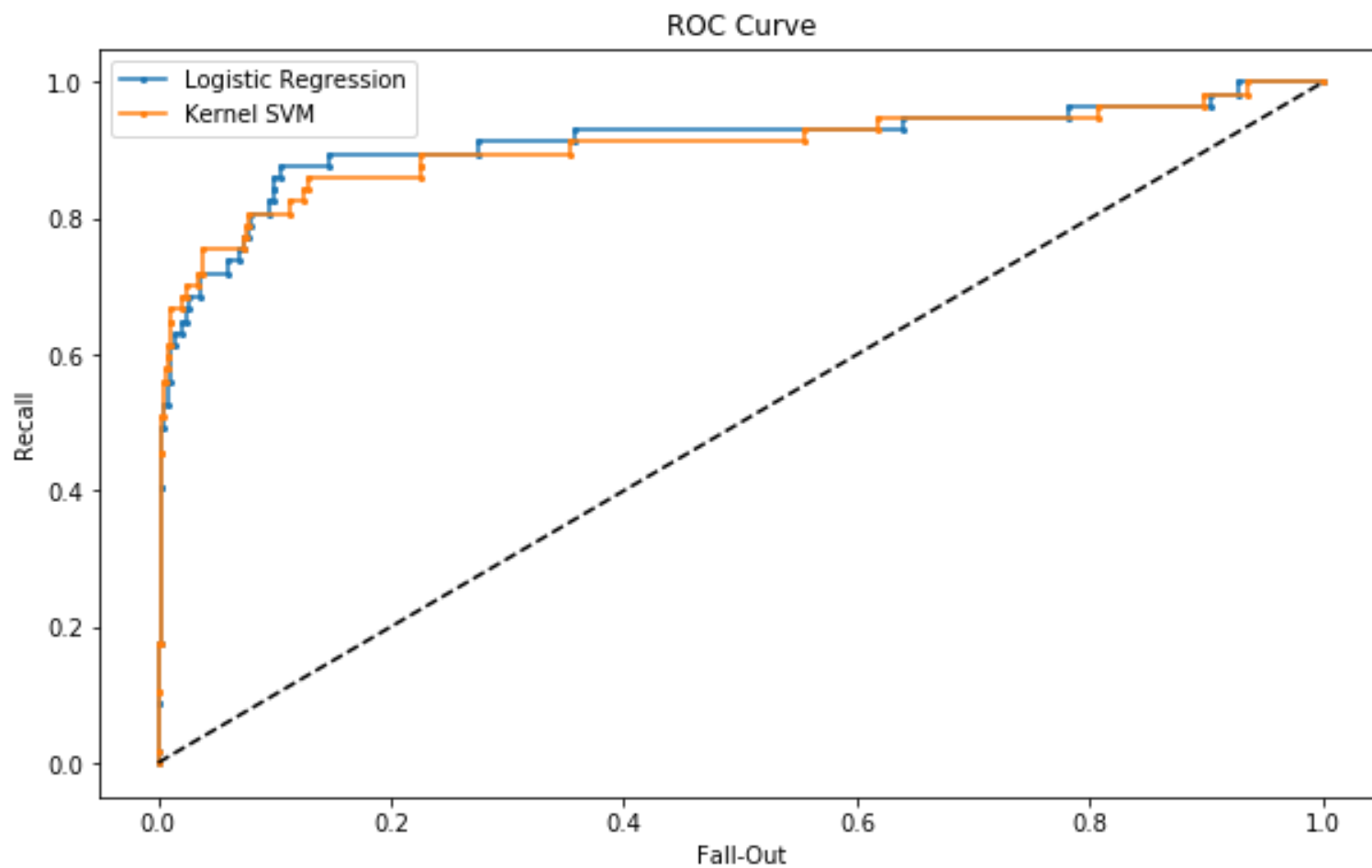
ROC 커브 (11)

```
plt.figure(figsize=(10,6))

fpr1, tpr1, thresholds1 = roc_curve(y, model1.decision_function(X))
fpr2, tpr2, thresholds1 = roc_curve(y, model2.decision_function(X))

plt.plot(fpr1, tpr1, 'o-', ms=2, label="Logistic Regression")
plt.plot(fpr2, tpr2, 'o-', ms=2, label="Kernel SVM")
plt.legend()
plt.plot([0, 1], [0, 1], 'k--', label="random guess")
plt.xlabel('Fall-Out')
plt.ylabel('Recall')
plt.title('ROC Curve')
plt.show()
```

ROC 커브 (12)



ROC 커브 (13)

AUC(Area Under the Curve)

- AUC는 ROC curve의 면적을 뜻한다.
- 위양성률값이 같을 때 재현률값이 크거나 재현률값이 같을 때 위양성률값이 작을수록
- AUC가 1에 가까운 값이고 좋은 모형이다.

```
▶ from sklearn.metrics import auc  
auc(fpr1, tpr1), auc(fpr2, tpr2)
```

20]: (0.9112202563673234, 0.9037227214377407)

ROC 커브 (14)

```
from sklearn.naive_bayes import GaussianNB
from sklearn.datasets import load_iris
from sklearn.preprocessing import label_binarize

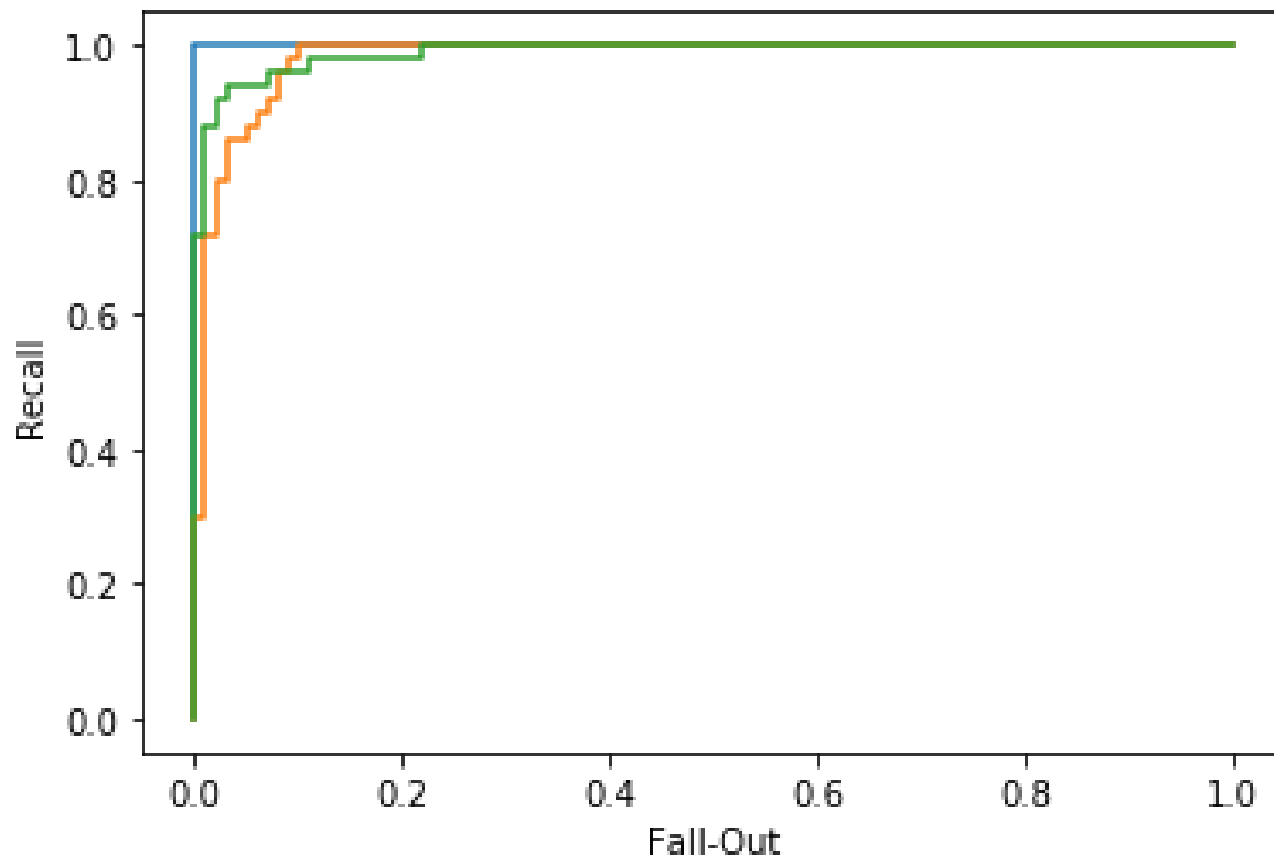
iris = load_iris()
X = iris.data
y = label_binarize(iris.target, [0, 1, 2])

fpr = [None] * 3; tpr = [None] * 3; thr = [None] * 3

for i in range(3):
    model = GaussianNB().fit(X, y[:, i])
    fpr[i], tpr[i], thr[i] = roc_curve(y[:, i], model.predict_proba(X)[:, 1])
    plt.plot(fpr[i], tpr[i])

plt.xlabel('Fall-Out')
plt.ylabel('Recall')
plt.show()
```

ROC 커브 (15)



타이타닉 데이터의 정확도 이해

- 정확도(Accuracy)

- ✓ 예측결과가 동일한 데이터 건수 / 전체 예측 데이터 건수
- ✓ 이진분류 문제에 주로 사용하며
- ✓ 정확도 1개만 이용하여 모델의 성능을 왜곡하여 평가 할 수 있음
 - 예제) 타이타닉의 여자 생존자를 레이블링
 - MNIST 데이터의 1개의 숫자를 분류하는 문제에서

- 장점

- ✓ 데이터가 균일할 때 좋다.

- 단점

- ✓ 데이터가 불균형한(imbalanced)일 경우 문제가 발생
- ✓ 오차행렬과 F1 스코어가 필요

혼동행렬 Confusion Matrix

		예측	
		Negative	Positive
실제	Negative	TN a[0,0]	FP a[0,1]
	Positive	FN a[1,0]	TP a[1,1]

3-2 Confusion Matrix

```
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)
```

정밀도와 재현율

- 정밀도 = $TP / (FP + TP)$
✓ FP를 낮추는 데 초점
- 재현율 = $TP / (FN + TP)$
✓ 재현율은 FN을 낮추는데 초점

		예측	
		Negative	Positive
실제	Negative	TN a[0,0]	FP a[0,1]
	Positive	FN a[1,0]	TP a[1,1]

정밀도 계산

- `precision_score()`와 `recall_score()` API 제공
- 편의를 위해 `get_clf_eval()` 함수 만듦
 - ✓ Confusion matrix, accuracy, precision, recall을 한꺼번에 호출

3-3 Precision과 Recall

```
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 계산

```
: 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('./input/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, W
                                                    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

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

정확도: 0.8492, 정밀도: 0.7742, 재현율: 0.7869

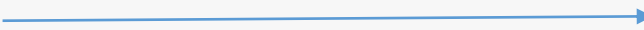
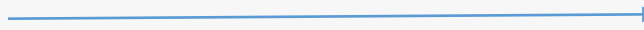
정밀도/재현율 트레이드오프

- 사이킷런 분류 알고리즘은 라벨(Label)을 결정 확률로 정함
 - ✓ (예) 임계값 50%로 두고, 51이상 확률이면 Positive, 이하면 Negative
 - ✓ `predict_proba()` 메서드 제공, 반환은 예측값이 아니고, 예측 확률을 제공.

정밀도/재현율 트레이드오프

Precision/Recall Trade-off

- 타이타닉의 로지스틱 회귀 객체에서 `predict_proba()`를 수행한 반환값


```
pred_proba = lr_clf.predict_proba(X_test)   LogisticRegression
pred = lr_clf.predict(X_test)               확률계산
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])
```

pred_proba()결과 Shape : (179, 2)

pred_proba array에서 앞 3개만 샘플로 추출

```
: [[0.46150057 0.53849943]
 [0.87861098 0.12138902]
 [0.87721519 0.12278481]]
```

 2개 컬럼 합하면 1.
합은 1
합은 1

두개의 class 중에서 더 큰 확률을 클래스 값으로 예측

```
[[0.46150057 0.53849943 1.]
 [0.87861098 0.12138902 0.]
 [0.87721519 0.12278481 0.]]
```

 예측한 라벨

임계값을 정하기 (1)

- 임계값에 따라, 즉 예측 확률에 따라 예측값이 다름.
- Binarizer 클래스 이용
 - ✓ 지정된 임계값 보다 작으면 0, 크면 1을 반환
 - ✓ fit_transform()을 사용

```
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.]]
```

-----> 임계값이 1.1이고, 이 값보다 크면 1을 출력

임계값을 정하기 (2)

- 디폴트 임계값=0.5일때 타이타닉의 같은 혼동행렬 얻어야 함

```
from sklearn.preprocessing import Binarizer

#Binarizer의 threshold 설정값. 분류 결정 임계값임.
custom_threshold = 0.5 -----> 디폴트 임계값을 사용

# 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)
```

혼동행렬

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

정확도: 0.8492, 정밀도: 0.7742, 재현율: 0.7869 -----> 같은 값인지 확인

임계값을 정하기 (3)

```
# Binarizer의 threshold 설정값을 0.4로 설정. 즉 분류 결정 임계값을 0.5에서 0.4로 낮춤
custom_threshold = 0.4 -----> 새로운 임계값
pred_proba_1 = pred_proba[:,1].reshape(-1,1) -----> Positive 확률
binarizer = Binarizer(threshold=custom_threshold).fit(pred_proba_1)
custom_predict = binarizer.transform(pred_proba_1)

get_clf_eval(y_test , custom_predict)
```

혼동행렬

[[99 19]

[10 51]]

정확도: 0.8380, 정밀도: 0.7286, 재현율: 0.8361

임계값=0.4

정밀도는 내려가고, 재현율은 올라간다.

혼동행렬

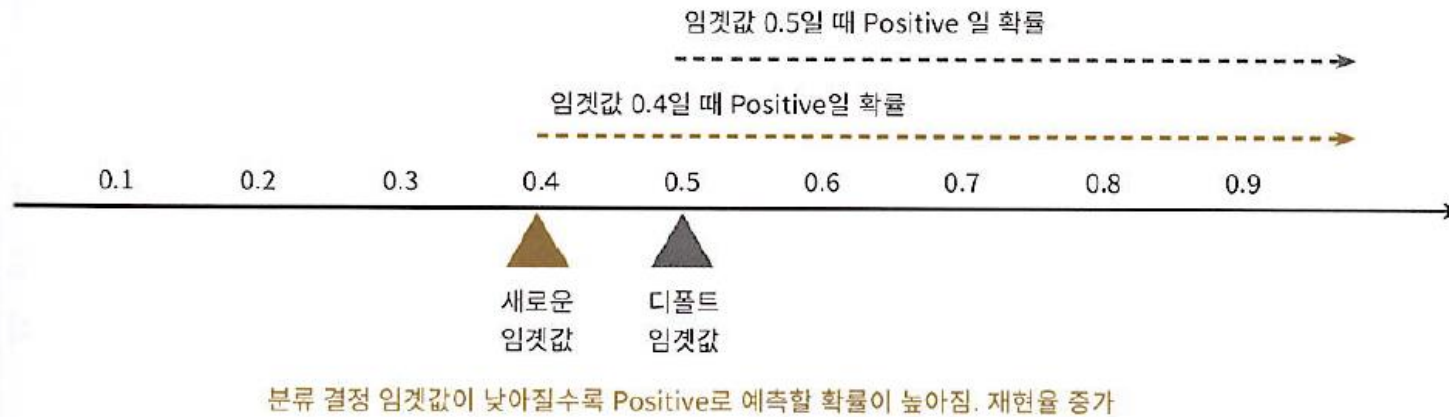
[[104 14]

[13 48]]

정확도: 0.8492, 정밀도: 0.7742, 재현율: 0.7869

임계값=0.5

임계값과 정밀도/재현율 (1)



임계값과 정밀도/재현율 (2)

```
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 임계값 배열의 Shape:', thresholds.shape)

#반환된 임계값 배열 로우가 147건이므로 샘플로 10건만 추출하되, 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 임계값: ', 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 임계값 배열의 Shape: (143,)

샘플 추출을 위한 임계값 배열의 index 10개: [0 15 30 45 60 75 90 105 120 135]

샘플용 10개의 임계값: [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]

임계값과 정밀도/재현율 (3)

```
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')

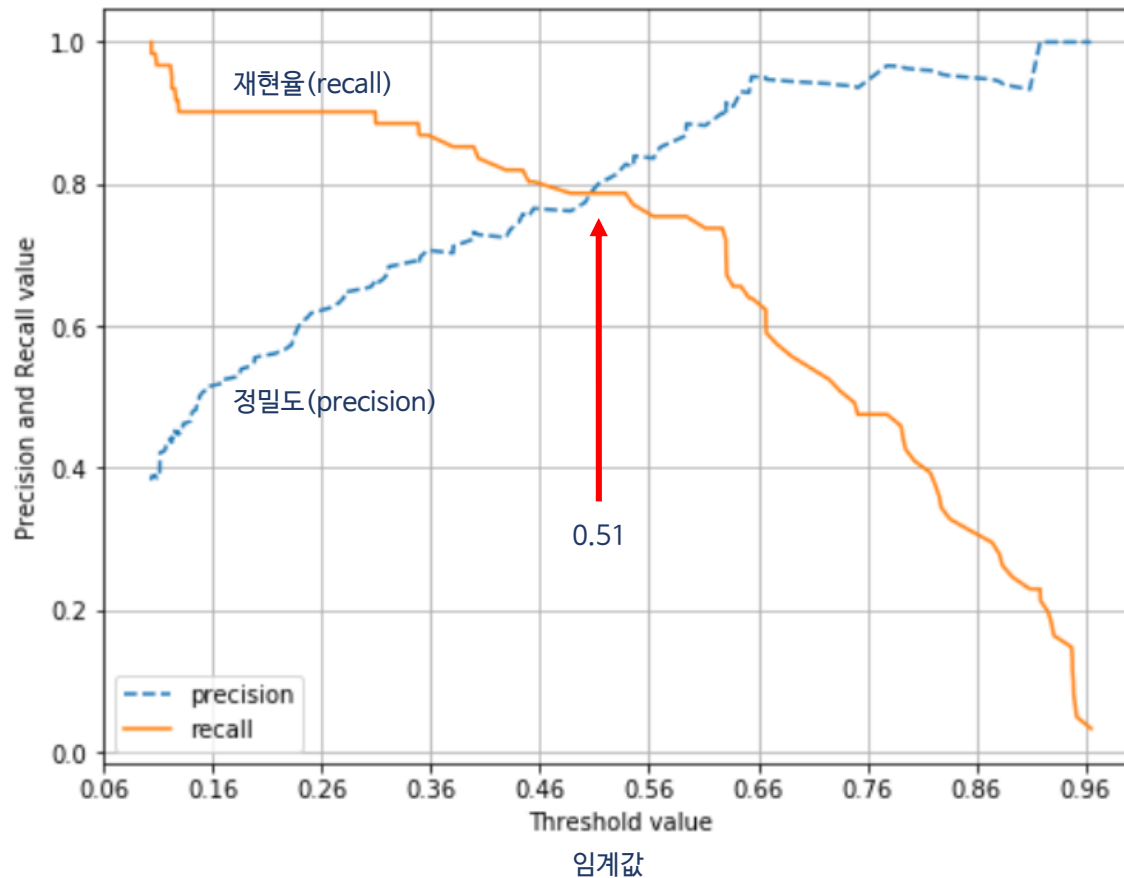
    # threshold 값 X 축의 Scale을 0.1 단위로 변경
    start, end = plt.xlim()
    plt.xticks(np.round(np.arange(start, end, 0.1),2))

    # 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] )
```

임계값과 정밀도/재현율 (4)

문제: 어떻게 정밀도와 재현율의 수치가 적절하게 조합돼 분류의 종합적인 성능평가에 사용할 것인가?

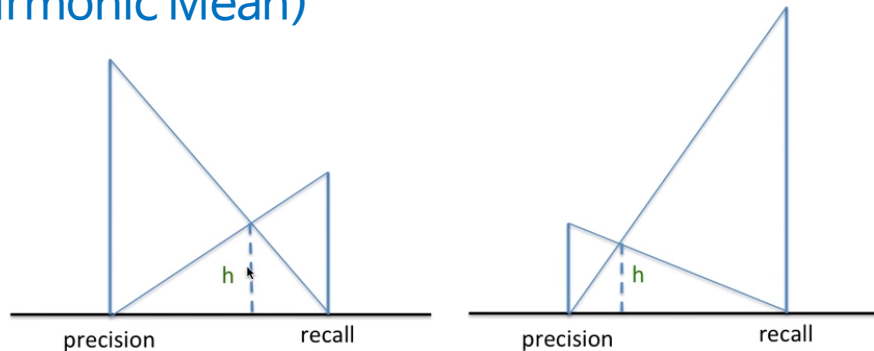


F1 스코어

- `f1_score()` API 제공
 - ✓ 정밀도와 재현율을 결합한 지표
 - ✓ 어느 한쪽으로 치우치지 않는 수치를 나타낼 때 상대적으로 높은 값을 가짐

$$F1 = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}}$$

조화평균(Harmonic Mean)



h is half the harmonic mean

$$F1 \text{ Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

F1 score

3.4 F1 Score

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

F1 스코어: 0.7805

F1 값이 최대일때, 임계값은 얼마인가?

```
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 = [0.3, 0.35, 0.4 , 0.45 , 0.50 , 0.55 , 0.60, 0.65, 0.7]
pred_proba = lr_clf.predict_proba(X_test)
get_eval_by_threshold(y_test, pred_proba[:,1].reshape(-1,1), thresholds)
```

오차 행렬

[[109 9]

[15 46]]

정확도: 0.8659, 정밀도: 0.8364, 재현율: 0.7541, F1:0.7931

임계값: 0.6

오차 행렬

[[112 6]

[16 45]]

정확도: 0.8771, 정밀도: 0.8824, 재현율: 0.7377, F1:0.8036

임계값: 0.65

오차 행렬

[[115 3]

[22 39]]

정확도: 0.8603, 정밀도: 0.9286, 재현율: 0.6393, F1:0.7573

임계값: 0.7

오차 행렬

[[116 2]

[28 33]]

정확도: 0.8324, 정밀도: 0.9429, 재현율: 0.5410, F1:0.6875

-----> 최대 F1

-----> 최대 임계값

ROC 커브와 AUC (1)

- ROC (Receiver Operation Characteristic Curve)
 - ✓ 수신자 판단 곡선
 - ✓ x 축: FPR(False Positive Rate)
 - TNR(True Negative Rate)은 특이성(Specificity)라고 부름
 - $FPR = FP / (FP + TN) = 1 - TNR = 1 - \text{특이성}$
 - ✓ y 축: TPR(True Positive Rate)
 - TPR은 recall(재현율)이다, 민감도(Sensitivity)라고 부름
 - ✓ ROC 곡선이 직선일 경우 (AUC=0.5)
 - 모델 예측 성능은 떨어짐
- 제공 API : `roc_curve()`

ROC 커브와 AUC (2)

- 분류의 판단 기준은 FPR과 TPR 변화를 면적 지표로 사용
- AUC (Area Under Curve)
 - ✓ 가운데 대각선 ROC 커브의 경우, 면적은 반. 즉 0.5이다.
 - ✓ AUC 면적이 1이면 성능이 좋다.

ROC 커브와 AUC (3)

문제1) 타이타닉 생존자 예측의 FPR, TPR, 임계값을 구하라

```
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_class1)
# 반환된 임계값 배열 로우가 47건이므로 샘플로 10건만 추출하되, 임계값을 5 Step으로 추출.
thr_index = np.arange(0, thresholds.shape[0], 5)
print('샘플 추출을 위한 임계값 배열의 index 10개:', thr_index)
print('샘플용 10개의 임계값: ', np.round(thresholds[thr_index], 2))

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

샘플 추출을 위한 임계값 배열의 index 10개: [0 5 10 15 20 25 30 35 40 45 50]
샘플용 10개의 임계값: [1.97 0.75 0.63 0.59 0.49 0.4 0.31 0.15 0.12 0.11 0.1]
샘플 임계값별 FPR: [0. 0.017 0.034 0.051 0.127 0.161 0.237 0.483 0.61 0.703 0.814]
샘플 임계값별 TPR: [0. 0.475 0.672 0.754 0.787 0.852 0.885 0.902 0.934 0.967 0.984]

ROC 커브와 AUC (4)

문제1) 타이타닉 생존자 예측의 FPR, TPR, 임계값을 구하라

```
from sklearn.metrics import roc_curve -----> ROC API

# 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_class1)
# 반환된 임계값 배열 로우가 47건이므로 샘플로 10건만 추출하되, 임계값을 5 Step으로 추출.
thr_index = np.arange(0, thresholds.shape[0], 5)
print('샘플 추출을 위한 임계값 배열의 index 10개:', thr_index)
print('샘플용 10개의 임계값: ', np.round(thresholds[thr_index], 2))

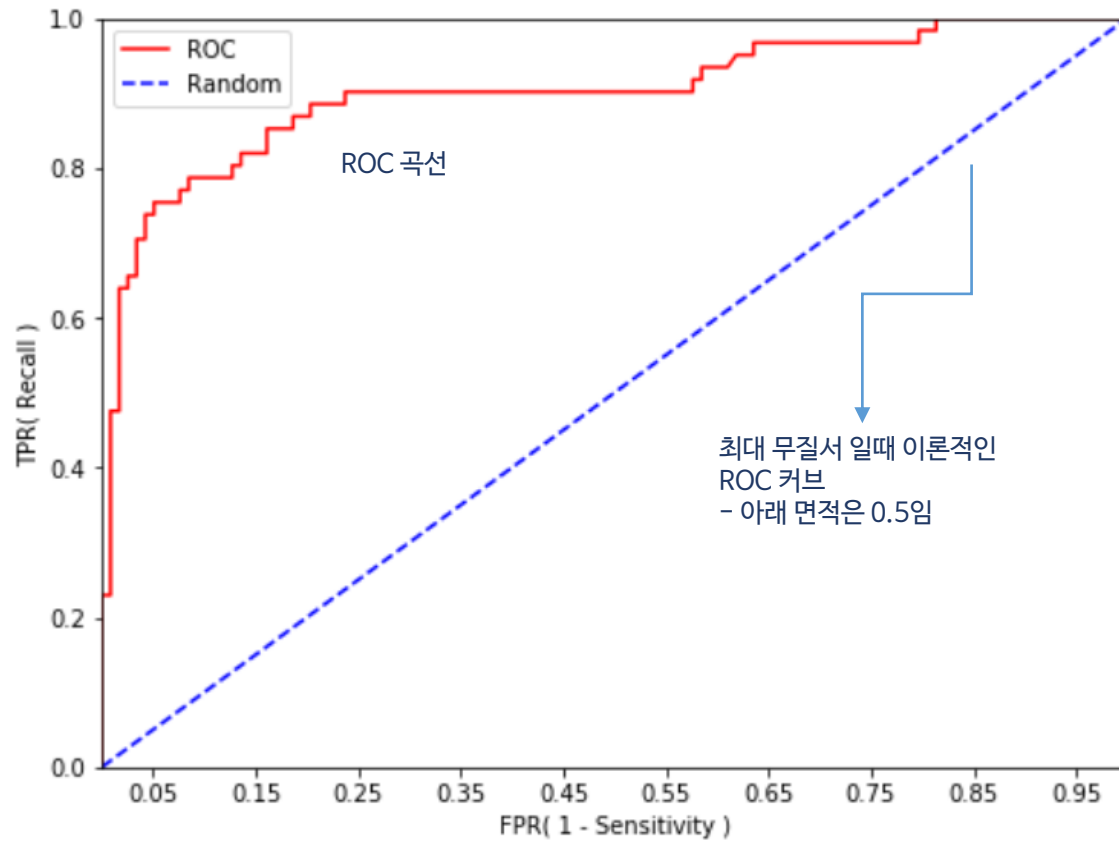
# 5 step 단위로 추출된 임계값에 따른 FPR, TPR 값
print('샘플 임계값별 FPR: ', np.round(fprs[thr_index], 3))
print('샘플 임계값별 TPR: ', np.round(tprs[thr_index], 3))
```

샘플 추출을 위한 임계값 배열의 index 10개: [0 5 10 15 20 25 30 35 40 45 50]
샘플용 10개의 임계값: [1.97 0.75 0.63 0.59 0.49 0.4 0.31 0.15 0.12 0.11 0.1]
샘플 임계값별 FPR: [0. 0.017 0.034 0.051 0.127 0.161 0.237 0.483 0.61 0.703 0.814]
샘플 임계값별 TPR: [0. 0.475 0.672 0.754 0.787 0.852 0.885 0.902 0.934 0.967 0.984]

ROC 커브와 AUC (5)

```
def roc_curve_plot(y_test , pred_proba_c1):  
    # 임계값에 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, 'r-', label='ROC')  
    # 가운데 대각선 직선을 그림.  
    plt.plot([0, 1], [0, 1], 'b--', 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 커브와 AUC (6)



ROC 커브와 AUC (7)

```
from sklearn.metrics import roc_auc_score

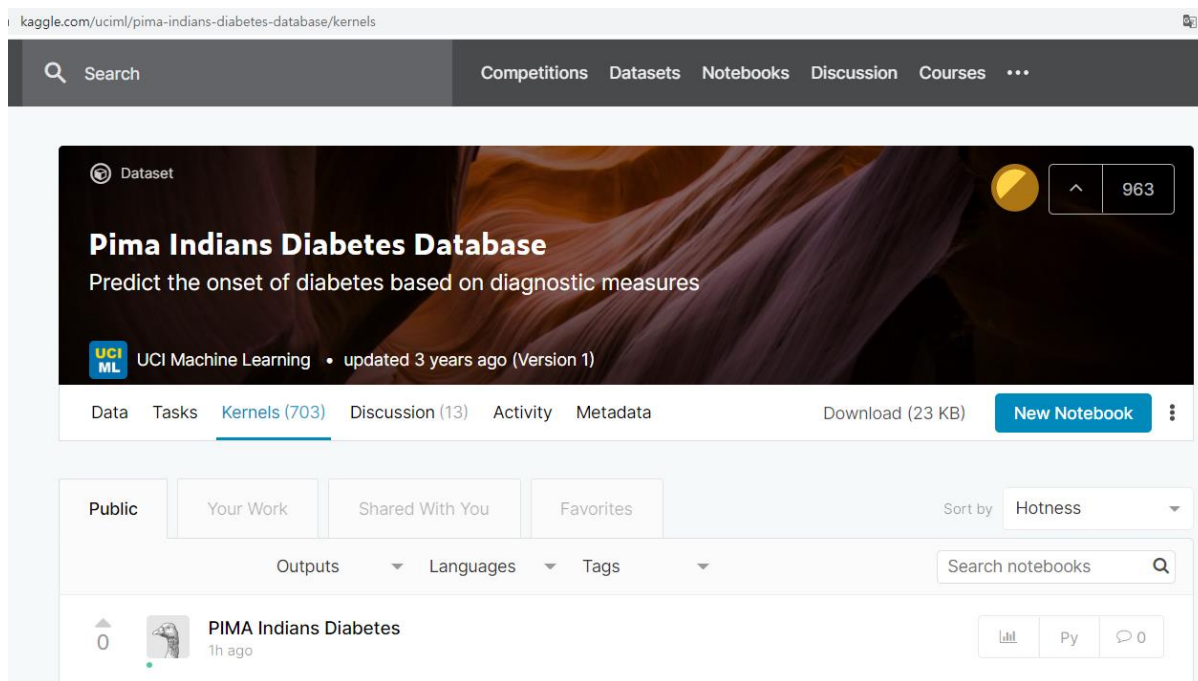
pred = lr_clf.predict(X_test)
roc_score = roc_auc_score(y_test, pred)
print('ROC AUC 값: {0:.4f}'.format(roc_score))
```

ROC AUC 값: 0.8341

```
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 = f1_score(y_test,pred)
    # ROC-AUC 추가
    roc_auc = roc_auc_score(y_test, pred)
    print('오차 행렬')
    print(confusion)
    # ROC-AUC print 추가
    print('정확도: {0:.4f}, 정밀도: {1:.4f}, 재현율: {2:.4f}, W
    F1: {3:.4f}, AUC:{4:.4f}'.format(accuracy, precision, recall, f1, roc_auc))
```

Kaggle – Pima 인디언 당뇨병 예측

- Kaggle에서 Pima 인디언 당뇨병 예측 코드와 데이터 받음
✓ Tutorial을 다운로드 받아본다.



(숙제) pima 인디언 당뇨병 예측(1)

3-6 (연습문제) 피마 인디언 당뇨병 예측

```
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('./input/diabetes.csv')
print(diabetes_data['Outcome'].value_counts())
```

-----> Kaggle에서 파일을 다운로드하고,
저장 위치를 input 폴더 안에 저장함.

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

(숙제) pima 인디언 당뇨병 예측(1)

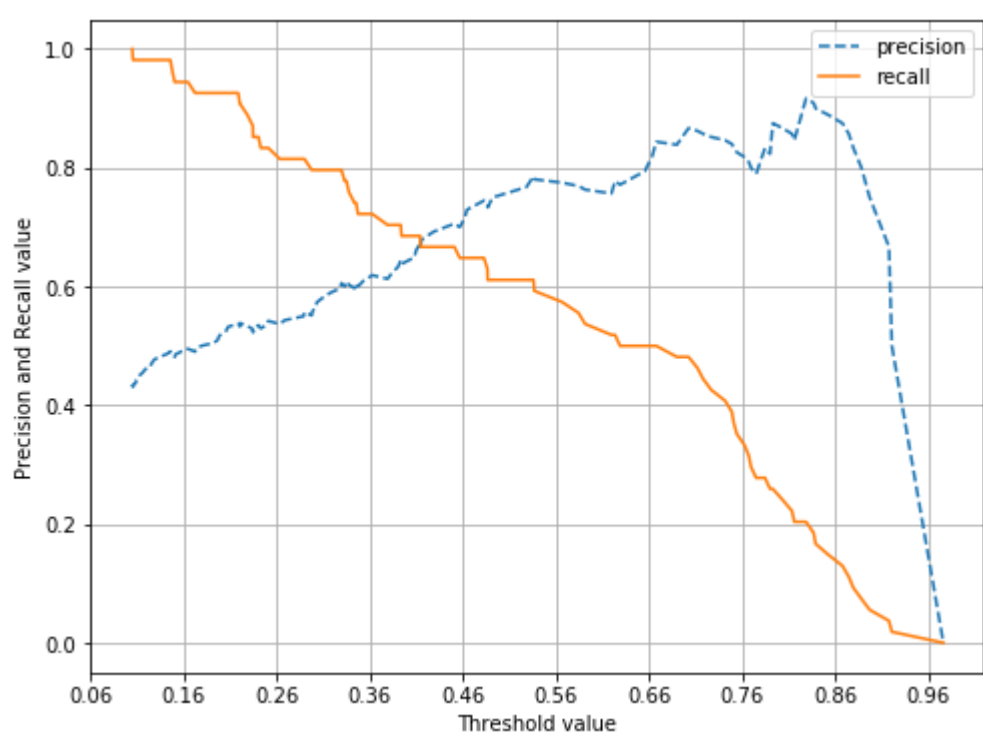
임신회수 포도당 부하 검사 수치 혈압(mm Hg) 피하지방 혈청 인슐린 체질량지수 당뇨 내력 가중치 값 나이 라벨

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148.0	72.0	35.000000	79.799479	33.6	0.627	50	1
1	1	85.0	66.0	29.000000	79.799479	26.6	0.351	31	0
2	8	183.0	64.0	20.536458	79.799479	23.3	0.672	32	1
3	1	89.0	66.0	23.000000	94.000000	28.1	0.167	21	0
4	0	137.0	40.0	35.000000	168.000000	43.1	2.288	33	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
```

(숙제) pima 인디언 당뇨병 예측(2)



3장 요약

- 이진 분류의 경우 데이터가 불균형일때,
 - ✓ 정확도 혼자서는 충분하지 않음
- 오차행렬, 정밀도, 재현율
 - ✓ 정밀도와 재현율(recall)은 Positive 데이터에 초점을 맞춘 평가지표
 - ✓ 임계밀도를 조정하여 정밀도와 재현율 수치를 제어
- F1 스코어
 - ✓ 정밀도와 재현율을 결합한 평가 지표이며, 어느 한쪽으로 치우치지 않을때 높은 값
- ROC-AUC 곡선
 - ✓ 일반적으로 이진 분류의 성능 평가로 가장 많이 사용하는 지표
 - ✓ AUC 값은 ROC 곡선 밑의 면적을 구한 것으로, 1에 가까울 수록 성능이 좋다.

Thank You!

www.ust.ac.kr