평가

[머신러닝 프로세스]

데이터 가공/변환 → 모델 학습/예측 → 평가

[**분류**]

결정 클래스 값 종류의 유형에 따라 **이진 분류**(ex.긍정/부정)와 **멀티 분류**로 나뉨

아래 성능 평가 지표는 이진/멀티 분류 모두에 적용 *이진 분류에서 더욱 강조

[**분류**의 성능 평가 지표]

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

*일반적으로 모델이 분류인지 회귀인지에 따라 성능 평가 지표는 여러 종류로 나뉨

*회귀 모델의 성능 평가 지표는 5장(링크: ___)

01 정확도(Accuracy)

: 실제 데이터에서 예측 데이터가 얼마나 같은지를 판단하는 지표

: 직관적인 모델 예측 성능 표현

*하지만 이진 분류의 경우 ML 모델의 성능을 왜곡할 가능성O → 정확도 외 다른 지표를 추가하여 성능 평가

정확도 = 예측 결과가 동일한 데이터 건수 / 전체 예측 데이터 건수

```
## 3-1 Accuracy(정확도)

source: [
    import numpy as np
    from sklearn.base import BaseEstimator

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

        # predict( ) 메소드는 단순히 Sex feature가 1 이면 0 , 그렇지 않으면 1 로 예측함.
        def predict(self, X):
            pred = np.zeros( ( X.shape[0], 1 ))
            for i in range (X.shape[0]) :
                if X['Sex'].iloc[i] == 1:
                    pred[i] = 0
                else :
                    pred[i] = 1

            return pred
   ]

source: [
    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)
```

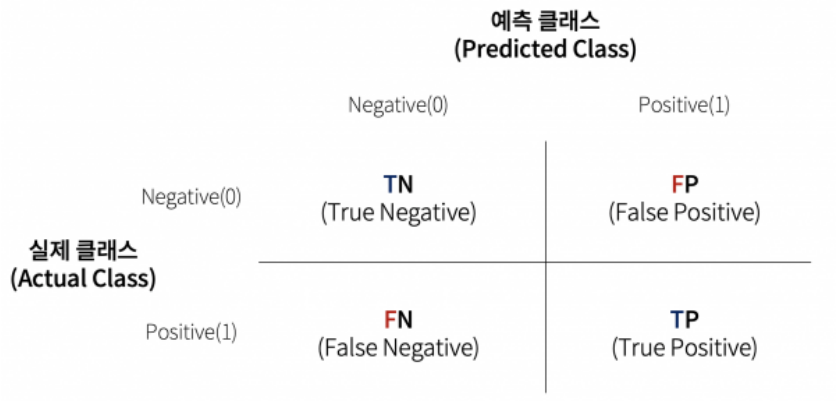
```
    X_train, X_test, y_train, y_test=train_test_split(X_titanic_df, y_titanic_df,
                                                      test_size=0.2, random_state=0)

    # 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: {0:.4f}'.format(accuracy_score(y_test , mypredictions)))
   ]

source: [
    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
        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()

source: [
    # 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)
   ]

source: [
    # 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)))
   ]
```

02 오차행렬

: 학습된 분류 모델이 예측을 수행하면서 얼마나 헷갈리고 있는지를 함께 보여주는 지표

: 이진 분류의 예측 오류 정도 & 발생하고 있는 예측 오류의 유형 표현

: True/False, Positive/Negative의 4분면으로 구성



- TN: 예측값 - Negative 값 0 예측, 실제 값 - Negative 값 0

- FP: 예측값 - Positive 값 1 예측, 실제 값 - Negative 값 0
- FN: 예측값 - Negative 값 0 예측, 실제 값 - Positive 값 1
- TP: 예측값 - Positive 값 1 예측, 실제 값 - Positive 값 1

정확도 = (TN + TP) / (TN +FP + FN + TP)

```
## Confusion Matrix

source: [
    from sklearn.metrics import confusion_matrix

    # 앞절의 예측 결과인 fakepred와 실제 결과인 y_test의 Confusion Matrix출력
    confusion_matrix(y_test , fakepred)
   ]
```

03 정밀도와 재현율

: Positive 데이터 세트의 예측 성능에 초점을 맞춘 평가 지표

: 불균형한 데이터 세트에서 정확도보다 선호되는 평가 지표

*앞서 만든 MyFakeClassifier은 Positive로 예측한 TP 값이 하나도 없기 때문에 정밀도와 재현율 값이 모두 0 임

정밀도 = TP / (FP + TP)

재현율 = TP / (FN + TP)

정밀도(=양성 예측도)

: 예측을 Positive로 한 대상 중에서 예측과 실제값이 Positive로 일치한 데이터의 비율

: 실제 Positive 양성인 데이터 예측을 Negative로 잘못 판단하게 되면 업무상 큰 영향이 발생하는 경우 ex. 암 판단 모델 - 오류의 대가 = 생명

: 사이킷런 제공 API - precision_score()

재현율(=민감도, TPR)

: 실제 값이 Positive인 대상 중에 예측과 실제 값이 Positive로 일치한 데이터의 비율

: 실제 Negative 음성인 데이터 예측을 Positive 양성으로 잘못 판단하게 되면 업무상 큰 영향이 발생하는 경우 ex. 스팸메일 여부 판단 모델 - 오류의 대가 = 업무 차질

: 사이킷런 제공 API - recall_score()

재현율과 정밀도 모두 높은 수치를 얻는 것이 가장 좋은 성능 평가

```
## 정밀도(Precision) 과 재현율(Recall)

source: [
    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))
   ]
```

```
source: [
    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)
   ]
```

정밀도/재현율 트레이드오프

: 정밀도와 재현율 중 어느 한쪽을 강제로 높이면 다른 하나의 수치가 떨어지는 현상

predic_proba()

:학습이 완료된 사이킷런 Classifier객체에서 호출 가능
: 테스트 피처 데이터 세트 입력 → 테스트 피처 레코드의 개별 클래스 예측 확률 반환(ndarray 형태)

```
### Precision/Recall Trade-off

source: [
    pred_proba = lr_clf.predict_proba(X_test)
    pred  = lr_clf.predict(X_test)
    print('pred_proba()결과 Shape : {0}'.format(pred_proba.shape))
    print('pred_proba array에서 앞 3개만 샘플로 추출 \\n:', pred_proba[:3])

    # 예측 확률 array 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])
   ]

source: [
    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))
   ]

source: [
    from sklearn.preprocessing import Binarizer

    #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)
    custom_predict = binarizer.transform(pred_proba_1)

    get_clf_eval(y_test, custom_predict)
   ]

source: [
```

```
    # Binarizer의 threshold 설정값을 0.4로 설정. 즉 분류 결정 임곗값을 0.5에서 0.4로 낮춤
    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)
   ]
```

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

    def get_eval_by_threshold(y_test , pred_proba_c1, thresholds):
        # thresholds list객체내의 값을 차례로 iteration하면서 Evaluation 수행.
        for custom_threshold in thresholds:
            binarizer = Binarizer(threshold=custom_threshold).fit(pred_proba_c1)
            custom_predict = binarizer.transform(pred_proba_c1)
            print('임곗값:',custom_threshold)
            get_clf_eval(y_test , custom_predict)

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

source: [
    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))
   ]

source: [
    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')\n",
        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] )
   ]
```

04 F1 스코어

: 정밀도와 재현율을 결합한 지표

: 정밀도와 재현율이 어느 한쪽으로 치우치지 않는 수치를 나타낼 때 높은 값을 가짐

F1 = 2*precision* recall / (precision+recall)

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

```
source: [
    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)
   ]
```

05 ROC 곡선과 AUC

: 이빈 분류의 예측 성능 측정에서 중요하게 사용되는 지표

ROC 곡선

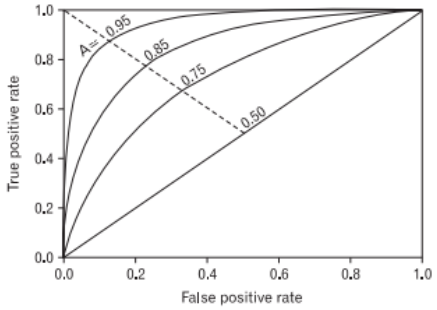
: FPR(x축)이 변할 때 TPR(y축)이 어떻게 변하는지를 나타내는 곡선

TPR = TP / (FN + TP) = 재현율

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

:사이킷런 제공 API - roc_curve() (p171)

ROC 곡선 예시



- **ROC 곡선이 가운데 직선에 가까울수록 성능이 떨어짐, 멀어질수록 성능이 뛰어남**
- FPR 을 0으로 만들려면 임계값을 1로 지정

- FPR을 1로 만들려면 TN을 0으로 만듦
- 임계값은 0~1까지 변화시키면서 FPR을 구하고 이 FPR 값의 변화에 따른 TPR값 구함

```
source: [
    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 임곗값 배열에서 샘플로 데이터를 추출하되, 임곗값을 5 Step으로 추출.
    # thresholds[0]은 max(예측확률)+1로 임의 설정됨. 이를 제외하기 위해 np.arange는 1부터 시작
    thr_index = np.arange(1, thresholds.shape[0], 5)
    print('샘플 추출을 위한 임곗값 배열의 index:', thr_index)
    print('샘플 index로 추출한 임곗값: ', np.round(thresholds[thr_index], 2))

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

source": [
    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] )
   ]

source: [
    from sklearn.metrics import roc_auc_score

    ### roc_auc_score(y_test, y_score)로 y_score는 predict_proba()로 호출된 예측 확률 ndarray중 Positive 열에 해당하는 ndarray입니다.

    #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: {0:.4f}'.format(roc_score))
   ]

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

06 피마 인디언 당뇨병 예측

- Pregnancies: 임신 횟수

- Glucose: 포도당 부하 검사 수치
- BloodPressure: 혈압(mm mg)
- SkinThickness: 삼두근 뒤쪽의 피하지방 측정값(mm)
- Insulin: 혈청인슐린(mu U/ml)
- BMI: 체질량지수(체중(kg)/키(m))^2)
- DiabetesPredigreeFunctinon: 당뇨 내력 가중치 값
- Age: 나이
- Outcome: 클래스 결정 값(0 or 1)

```
source: [
    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('diabetes.csv')
    print(diabetes_data['Outcome'].value_counts())
    diabetes_data.head(3)
   ]

source: [
    diabetes_data.info( )
   ]

source: [
    # 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)
   ]

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

source: [
    diabetes_data.describe()
   ]

source: [
    plt.hist(diabetes_data['Glucose'], bins=10)
   ]

source: [
    # 0값을 검사할 피처명 리스트 객체 설정
    zero_features = ['Glucose', 'BloodPressure','SkinThickness','Insulin','BMI']

    # 전체 데이터 건수
    total_count = diabetes_data['Glucose'].count()

    # 피처별로 반복 하면서 데이터 값이 0 인 데이터 건수 추출하고, 퍼센트 계산
    for feature in zero_features:
        zero_count = diabetes_data[diabetes_data[feature] == 0][feature].count()
        print('{0} 0 건수는 {1}, 퍼센트는 {2:.2f} %'.format(feature, zero_count, 100*zero_count/total_count))
   ]
```

```
source: [
    # zero_features 리스트 내부에 저장된 개별 피처들에 대해서 0값을 평균 값으로 대체
    diabetes_data[zero_features]=diabetes_data[zero_features].replace(0, diabetes_data[zero_features].mean())
   ]

source: [
    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)
   ]


source: [
    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)
   ]

source: [
    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 )
   ]

source: [
    # 임곗값를 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])
   ]
```