

4th Meet

□ 오늘 공부할 것은

03 평가	01. 정확도(Accuracy)	146
	02. 오차 행렬	150
	03. 정밀도와 재현율	154
	정밀도/재현율 트레이드오프	157
	정밀도와 재현율의 명점	165
	04. F1 스코어	166
	05. ROC 곡선과 AUC	167
	06. 피마 인디언 당뇨병 예측	172
	07. 정리	180

Now, It's your turn ! ! !

40min ~ 50min

Chapter 03

평가

[03-01] 정확도 (Accuracy)

[03-02] 오차행렬 (Confusion Matrix)

[03-03] 정밀도와 재현율 (Precision & Recall)

[03-04] F1 스코어 (F1 Score)

[03-05] ROC 곡선과 AUC

- ROC Curve : Receiver Operation Characteristic Curve
- AUC Score : Area Under Curve Score

[03-06] 피마 인디언 당뇨병 예측

[03-07] 정리

[03-01]

정확도 (Accuracy)

□ 정확도의 성능 왜곡

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

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder

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

def drop_features(df):
    df.drop(['PassengerId', 'Name', 'Ticket'], axis=1, inplace=True)
    return df

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

def transform_features(df):
    df = fillna(df)
    df = drop_features(df)
    df = format_features(df)
    return df

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

df = pd.read_csv('./titanic_train.csv')

y_df = df['Survived']
X_df = df.drop('Survived', axis=1)
X_df = transform_features(X_df)

X_train, X_test, y_train, y_test = train_test_split(X_df, y_df, test_size=0.2, random_state=0)
```

```
import numpy as np
from sklearn.base import BaseEstimator

class MyDummyClassifier(BaseEstimator):

    def fit(self, X, y=None):
        pass

    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

myclf = MyDummyClassifier()
myclf.fit(X_train, y_train)

mypredictions = myclf.predict(X_test)
accuracy_score(y_test, mypredictions)

0.7877094972067039
```

'여성'의 생존율만으로 정확도 높이기

□ 불균형 레이블 (imbalanced label)

```
from sklearn.datasets import load_digits
digits = load_digits()
digits.data.shape, digits.target.shape
((1797, 64), (1797,))

digits.target
array([0, 1, 2, ..., 8, 9, 8])

y = (digits.target == 7).astype(int)
y
array([0, 0, 0, ..., 0, 0, 0])

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(digits.data, y, test_size=0.2, random_state=11)
y_train.shape, y_test.shape
((1437,), (360,))

import pandas as pd
pd.Series(y_test).value_counts()
0    324
1     36
Name: count, dtype: int64
```

0~9 중에서 7을 맞추기

→ 무조건 False라고 해도 90%

```
from sklearn.base import BaseEstimator
from sklearn.metrics import accuracy_score

class MyFakeClassifier(BaseEstimator):
    def fit(self, X, y):
        pass

    def predict(self, X):
        return np.zeros((len(X), 1), dtype=bool)

fake_clf = MyFakeClassifier()
fake_clf.fit(X_train, y_train)

fake_pred = fake_clf.predict(X_test)
accuracy_score(y_test, fake_pred)

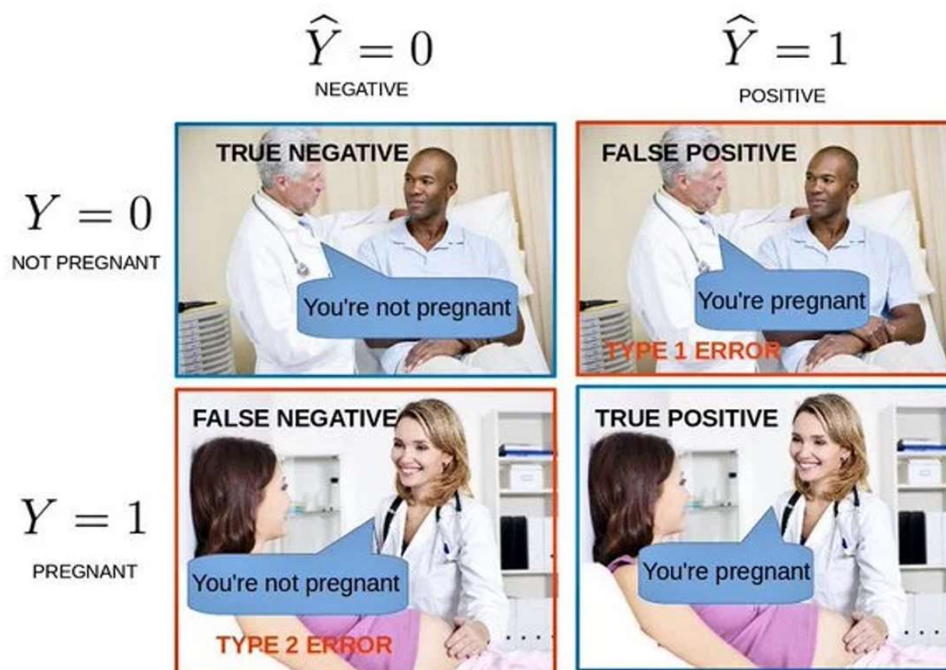
0.9
```


[03-02]

오차 행렬 (Confusion Matrix)

□ Confusion Matrix

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



□ Confusion Matrix

```
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

digits = load_digits()

y = (digits.target == 7).astype(int)
X_train, X_test, y_train, y_test = train_test_split(digits.data, y, test_size=0.2, random_state=11)

class MyFakeClassifier(BaseEstimator):
    def fit(self, X, y):
        pass

    def predict(self, X):
        return np.zeros((len(X), 1), dtype=bool)

fake_clf = MyFakeClassifier()
fake_clf.fit(X_train, y_train)

fake_pred = fake_clf.predict(X_test)
accuracy_score(y_test, fake_pred)

0.9
```

```
from sklearn.metrics import confusion_matrix

confusion_matrix(y_test, fake_pred)

array([[324,  0],
       [ 36,  0]])
```

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

7이 아니라고 예측했는데, 정말 아닌 경우 324건

7이 아니라고 예측했는데, 7인 경우 36건

□ Confusion Matrix

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

$$\text{정확도 (Accuracy)} = \frac{\text{예측 결과가 동일한 데이터 건수}}{\text{전체 예측 데이터 건수}} = \frac{TN + TP}{TN + FP + FN + TP}$$

[03-03]

정밀도와 재현율 (Precision & Recall)

□ 정밀도와 재현율 (Precision & Recall)

Positive로 예측한 것 중에서

정말 Positive한 데이터 비율

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

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

실제 Positive로 데이터 중에서

Positive로 예측한 비율 = 민감도 (Sensitivity)

= TPR (True Positive Rate)

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

□ 정밀도와 재현율 (Precision & Recall)

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split

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

def drop_features(df):
    df.drop(['PassengerId', 'Name', 'Ticket'], axis=1, inplace=True)
    return df

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

def transform_features(df):
    df = fillna(df)
    df = drop_features(df)
    df = format_features(df)
    return df

df = pd.read_csv('./titanic_train.csv')

y_df = df['Survived']
X_df = df.drop('Survived', axis=1)
X_df = transform_features(X_df)

X_train, X_test, y_train, y_test = train_test_split(X_df, y_df,
                                                    test_size=0.20,
                                                    random_state=11)
```

```
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))
```

```
from sklearn.linear_model import LogisticRegression
```

```
lr_clf = LogisticRegression(max_iter=200)
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

□ Trade-Off - predict_proba()

predict_proba()의 확률을 이용해서

Precision과 Recall 조정 가능 → 분류 결정 임계값 (threshold)

```
pred_proba = lr_clf.predict_proba(X_test)

pred_proba[:3]

array([[0.4622624 , 0.5377376 ],
       [0.87876959, 0.12123041],
       [0.87722282, 0.12277718]])

pred.reshape(-1,1)[:3]

array([[1],
       [0],
       [0]])

import numpy as np

pred_proba_result = np.concatenate([pred_proba , pred.reshape(-1,1)],axis=1)
pred_proba_result[:3]

array([[0.4622624 , 0.5377376 , 1.        ],
       [0.87876959, 0.12123041, 0.        ],
       [0.87722282, 0.12277718, 0.        ]])
```


□ Trade-Off - Binarizer

```
from sklearn.preprocessing import Binarizer

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

binarizer = Binarizer(threshold=1.1)
bin = binarizer.fit_transform(X)
bin
```

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

임계 값을 기준으로 결과를 분류

여기에서는 predict_proba()의

positive 항목을 기준으로

분류하도록 구현

Accuracy vs. Precision vs. Recall

```
pred_proba_1 = pred_proba[:,1].reshape(-1,1)
pred_proba_1[:3]

array([[0.5377376 ],
       [0.12123041],
       [0.12277718]])

custom_threshold = 0.5

binarizer = Binarizer(threshold=custom_threshold)
custom_predict = binarizer.fit_transform(pred_proba_1)
custom_predict[:3]

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

get_clf_eval(y_test, custom_predict)

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

custom_threshold = 0.4

binarizer = Binarizer(threshold=custom_threshold)
custom_predict = binarizer.fit_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
```

```
thresholds = [0.4, 0.45, 0.50, 0.55, 0.60]

def get_eval_by_threshold(y_test, pred_proba_c1, thresholds):

    for custom_threshold in thresholds:
        binarizer = Binarizer(threshold=custom_threshold)
        custom_predict = binarizer.fit_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
[[98 20]
 [10 51]]
정확도: 0.8324, 정밀도: 0.7183, 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

□ Trade-Off - precision_recall_curve()

```
from sklearn.metrics import precision_recall_curve

lr_clf = LogisticRegression(max_iter=200)
lr_clf.fit(X_train, y_train)

pred_proba_class1 = lr_clf.predict_proba(X_test)[:, 1]
pred_proba_class1[:5]

array([0.5377376, 0.12123041, 0.12277718, 0.11755966, 0.14473053])
```

```
precisions, recalls, thresholds = precision_recall_curve(y_test, pred_proba_class1)
```

```
print('반환된 분류 결정 임계값 배열의 Shape:', thresholds.shape)
print('반환된 precisions 배열의 Shape:', precisions.shape)
print('반환된 recalls 배열의 Shape:', recalls.shape)
print()
print("thresholds 5 sample:", thresholds[:5])
print("precisions 5 sample:", precisions[:5])
print("recalls 5 sample:", recalls[:5])
print()
thr_index = np.arange(0, thresholds.shape[0], 15)
print('샘플 추출을 위한 임계값 배열의 index 11개:', thr_index)
print('샘플용 11개의 임계값: ', np.round(thresholds[thr_index], 2))
print()
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: (165,)
반환된 precisions 배열의 Shape: (166,)
반환된 recalls 배열의 Shape: (166,)
```

```
thresholds 5 sample: [0.01157964 0.05285578 0.06228381 0.06364646 0.06863153]
precisions 5 sample: [0.34078212 0.34269663 0.34463277 0.34659091 0.34857143]
recalls 5 sample: [1. 1. 1. 1. 1.]
```

```
샘플 추출을 위한 임계값 배열의 index 11개: [ 0 15 30 45 60 75 90 105 120 135 150]
샘플용 11개의 임계값: [0.01 0.09 0.11 0.13 0.15 0.23 0.35 0.5 0.63 0.75 0.89]
```

```
샘플 임계값별 정밀도: [0.341 0.372 0.415 0.448 0.505 0.585 0.688 0.774 0.913 0.935 0.938]
샘플 임계값별 재현율: [1. 1. 0.967 0.918 0.902 0.902 0.869 0.787 0.689 0.475 0.246]
```

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

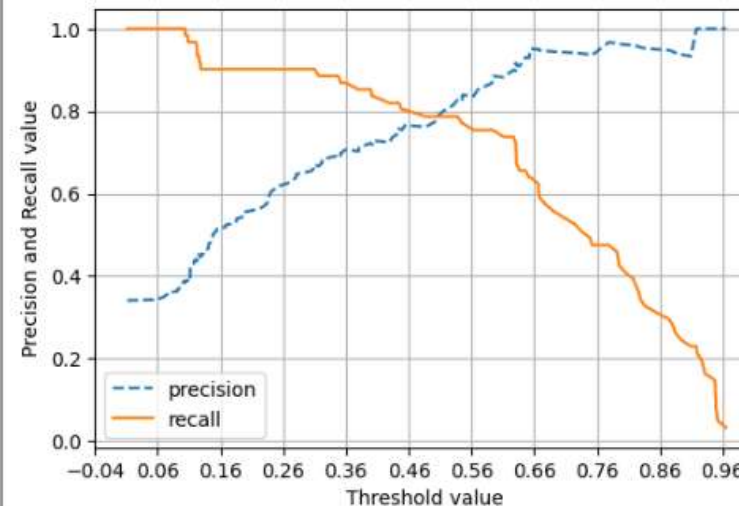
def precision_recall_curve_plot(y_test, pred_proba_c1):
    precisions, recalls, thresholds = precision_recall_curve(y_test, pred_proba_c1)

    plt.figure(figsize=(6,4))
    threshold_boundary = thresholds.shape[0]
    plt.plot(thresholds, precisions[0:threshold_boundary], linestyle='--', label='precision')
    plt.plot(thresholds, recalls[0:threshold_boundary], label='recall')

    start, end = plt.xlim()
    plt.xticks(np.round(np.arange(start, end, 0.1), 2))

    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])
```



[03-04]

F1 스코어 (F1 Score)

□ F1 Score

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

precision과 recall의 조화평균

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, confusion_matrix, f1_score

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)

    print('오차 행렬')
    print(confusion)
    print('정확도: {0:.4f}, 정밀도: {1:.4f}, 재현율: {2:.4f}, F1:{3:.4f}'.format(accuracy, precision, recall, f1))

from sklearn.preprocessing import Binarizer

def get_eval_by_threshold(y_test, pred_proba_c1, thresholds):

    for custom_threshold in thresholds:
        binarizer = Binarizer(threshold=custom_threshold)
        custom_predict = binarizer.fit_transform(pred_proba_c1)

        print('임계값:', custom_threshold)
        get_clf_eval(y_test, custom_predict)
        print()
```

```
pred_proba = lr_clf.predict_proba(X_test)

thresholds = [0.4, 0.45, 0.50, 0.55, 0.60]
get_eval_by_threshold(y_test, pred_proba[:,1].reshape(-1,1), thresholds)

임계값: 0.4
오차 행렬
[[98 20]
 [10 51]]
정확도: 0.8324, 정밀도: 0.7183, 재현율: 0.8361, F1:0.7727

임계값: 0.45
오차 행렬
[[103 15]
 [ 12 49]]
정확도: 0.8492, 정밀도: 0.7656, 재현율: 0.8033, F1:0.7840

임계값: 0.5
오차 행렬
[[104 14]
 [ 13 48]]
정확도: 0.8492, 정밀도: 0.7742, 재현율: 0.7869, F1:0.7805

임계값: 0.55
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
```

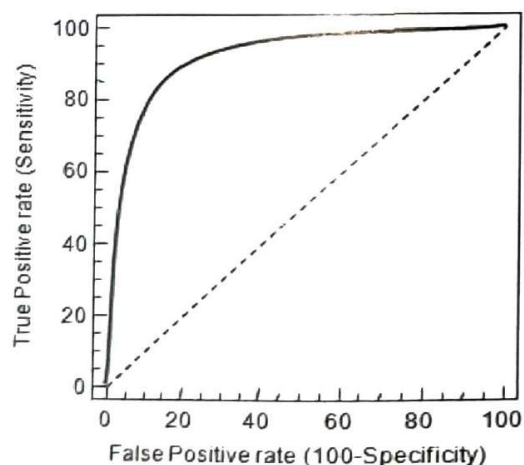
[03-05]

ROC 곡선과 AUC

(ROC Curve and AUC Score)

□ ROC 곡선 (Receiver Operation Characteristic Curve, 수신자 판단 곡선)

- FPR(False Positive Rate)이 변할 때 TPR(True Positive Rate)이 어떻게 변하는지를 표현



〈 ROC 곡선 예시 〉

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

$$\text{재현율 (Recall)} = \text{TPR(민감도, 양성률)} = \frac{TP}{FN + TP}$$

True Positive Ratio (Sensitivity)

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

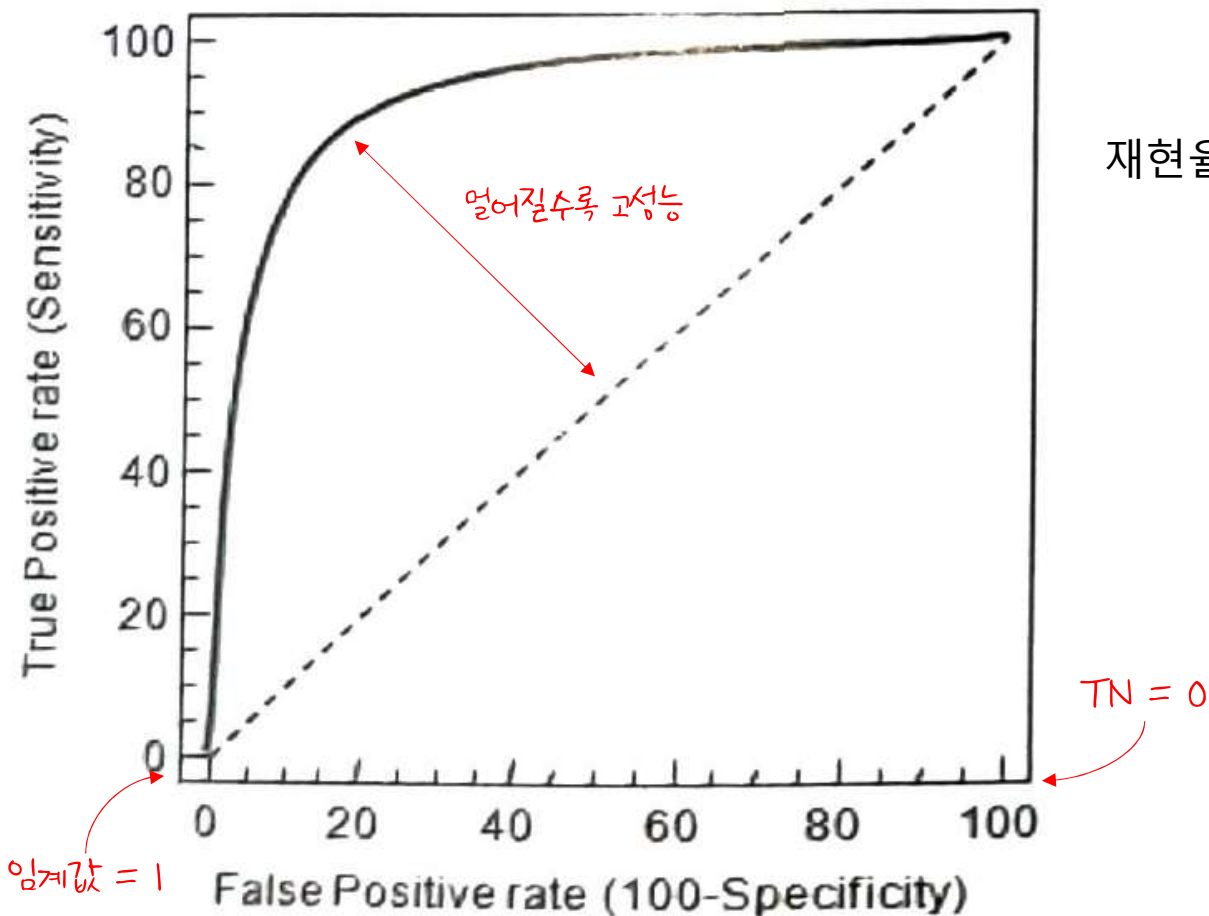
True Negative Ratio (Specificity)

$$\text{FPR(위양성률)} = \frac{FP}{FP + TN}$$

False Positive Ratio

$$= 1 - \text{TNR}$$

□ ROC 곡선 (Receiver Operation Characteristic Curve, 수신자 판단 곡선)



〈 ROC 곡선 예시 〉

$$\text{재현율 (Recall)} = \text{TPR}(\text{민감도}) = \frac{TP}{FN + TP}$$

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

$$\begin{aligned} \text{FPR} &= \frac{FP}{FP + TN} \\ &= 1 - \text{TNR}(\text{특이성}) \end{aligned}$$

분류 결정 임계값(threshold) 변경을 통해 FPR 변경

□ ROC 곡선 - roc_curve

```
import numpy as np
from sklearn.metrics import roc_curve

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])

thr_index = np.arange(1, thresholds.shape[0], 5)
print()
print('샘플 추출을 위한 임계값 배열의 index:', thr_index)
print('샘플 index로 추출한 임계값: ', np.round(thresholds[thr_index], 2))
print()
print('샘플 임계값별 FPR: ', np.round(fprs[thr_index], 3))
print('샘플 임계값별 TPR: ', np.round(tprs[thr_index], 3))

max predict_proba: 0.9650575313348472
thresholds[0]: inf

샘플 추출을 위한 임계값 배열의 index: [ 1  6 11 16 21 26 31 36 41 46 51]
샘플 index로 추출한 임계값: [0.97 0.65 0.63 0.56 0.45 0.4  0.35 0.15 0.13 0.11 0.11]

샘플 임계값별 FPR: [0.    0.017 0.034 0.076 0.127 0.169 0.203 0.466 0.585 0.686 0.797]
샘플 임계값별 TPR: [0.033 0.639 0.721 0.754 0.803 0.836 0.885 0.902 0.934 0.967 0.984]
```

```
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])

thr_index = np.arange(0, thresholds.shape[0], 5)
print()
print('샘플 추출을 위한 임계값 배열의 index 10개:', thr_index)
print('샘플용 10개의 임계값: ', np.round(thresholds[thr_index], 2))
print()
print('샘플 임계값별 FPR: ', np.round(fprs[thr_index], 3))
print('샘플 임계값별 TPR: ', np.round(tprs[thr_index], 3))

max predict_proba: 0.9650575313348472
thresholds[0]: inf

샘플 추출을 위한 임계값 배열의 index 10개: [ 0  5 10 15 20 25 30 35 40 45 50]
샘플용 10개의 임계값: [ inf 0.75 0.63 0.59 0.49 0.4  0.35 0.23 0.13 0.12 0.11]

샘플 임계값별 FPR: [0.    0.017 0.034 0.051 0.127 0.161 0.203 0.331 0.585 0.636 0.797]
샘플 임계값별 TPR: [0.    0.475 0.689 0.754 0.787 0.836 0.869 0.902 0.918 0.967 0.967]
```


□ ROC 곡선 - roc_curve

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

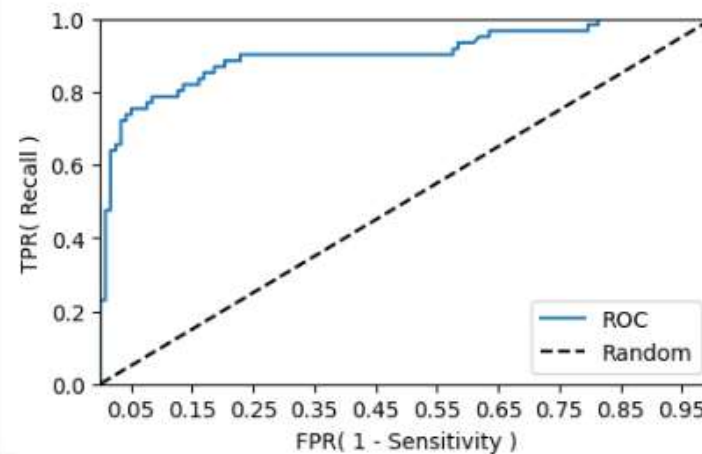
def roc_curve_plot(y_test , pred_proba_c1):

    fprs , tprs , thresholds = roc_curve(y_test ,pred_proba_c1)

    plt.figure(figsize=(5,3))
    plt.plot(fprs , tprs, label='ROC')
    plt.plot([0, 1], [0, 1], 'k--', label='Random')

    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] )
```



[03-06]

피마 인디언 당뇨병 예측

(Pima Indians Diabetes)

□ Dataset

미국 애리조나주 피마 인디언과 멕시코 피마 인디언은 유전자가 같은 형제 부족이다. 멕시코 피마 인디언은 여전히 '몸짱'을 자랑하며 날렵하고 건강한 모습으로 생활하고 있지만 예전에 강인한 체력을 자랑했던 애리조나 주 피마 인디언은 부족의 70%가 당뇨를 앓고 있는 세계 최악의 '당뇨병 부족'이라는 오명을 안고 살아가고 있다. 그들이 특별히 당뇨에 취약한 유전자를 물려받은 건 아닐까? 그렇다면 멕시코 피마 인디언은 왜 당뇨, 암, 심장병이 없이 건강히 살아가고 있는 걸까?



<https://www.britannica.com/topic/Pima-people>



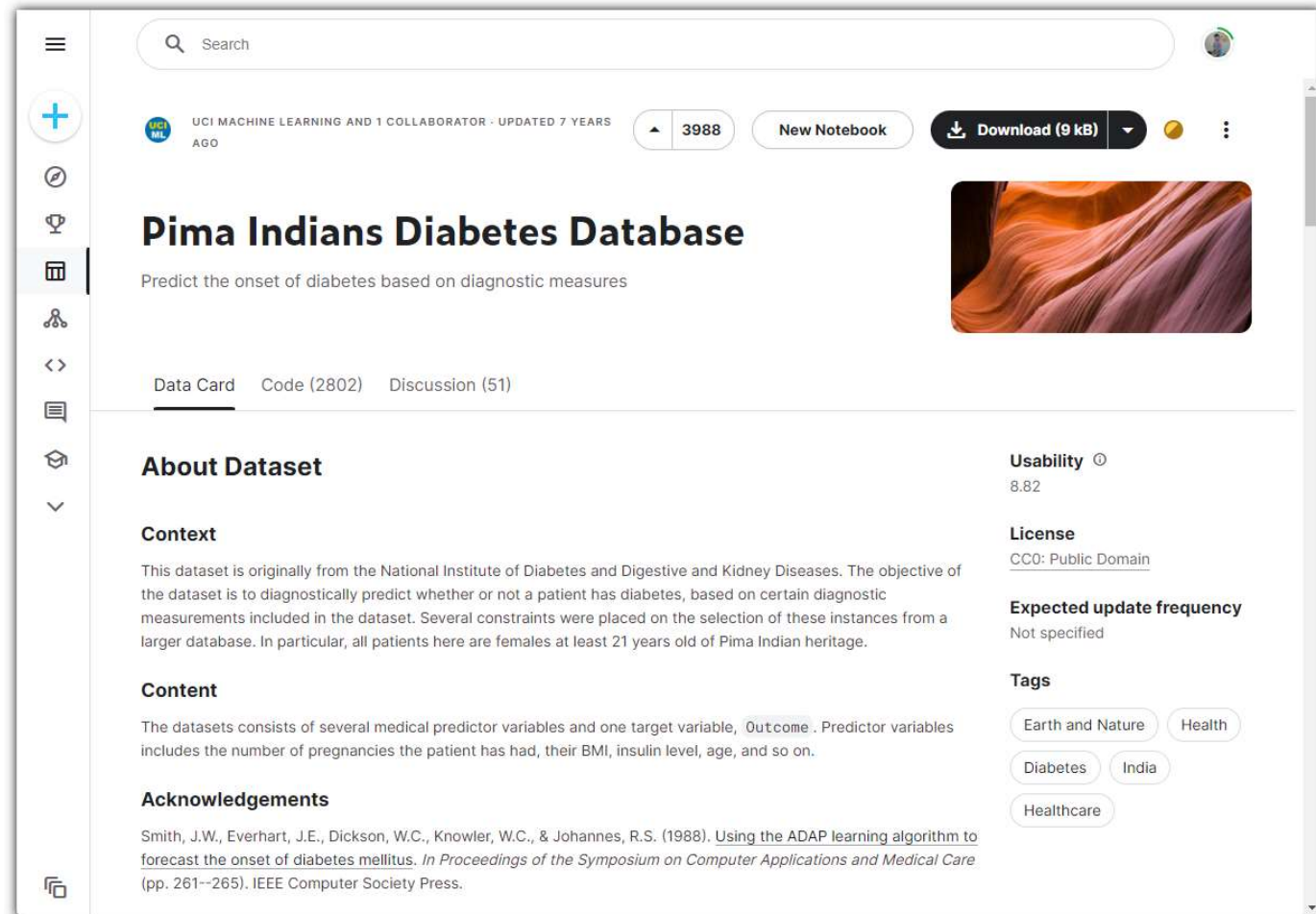
<http://www.historyadventuring.com/2015/06/the-pima-indians-living-in-desert-for.html>

※ 출처 : <https://www.jejusori.net/news/articleView.html?idxno=106341>

□ Dataset

북아메리카 피마 지역 원주민의
당뇨병 결과 데이터

Pregnancies	임신 횟수
Glucose	포도당 부하 검사 수치
BloodPressure	혈압 (mm Hg)
SkinThickness	팔 삼두근 뒤쪽 피하지방 (mm)
Insulin	혈청 인슐린 (mu U/ml)
BMI	체질량 지수
DiabetesPedigreeFunction	당뇨 내력 가중치 값
Age	나이
Outcome	클래스 결정 값 (0 또는 1)



The screenshot shows the Kaggle dataset page for the Pima Indians Diabetes Database. The page includes a search bar, a sidebar with navigation icons, and a main content area. The main content area has a title 'Pima Indians Diabetes Database' and a subtitle 'Predict the onset of diabetes based on diagnostic measures'. Below the title, there are tabs for 'Data Card', 'Code (2802)', and 'Discussion (51)'. The 'Data Card' tab is selected, showing the 'About Dataset' section. The 'About Dataset' section includes a 'Context' paragraph, a 'Content' paragraph, and an 'Acknowledgements' section. On the right side, there are sections for 'Usability' (8.82), 'License' (CC0: Public Domain), 'Expected update frequency' (Not specified), and 'Tags' (Earth and Nature, Health, Diabetes, India, Healthcare).

UCI MACHINE LEARNING AND 1 COLLABORATOR · UPDATED 7 YEARS AGO

3988 New Notebook Download (9 kB)

Pima Indians Diabetes Database

Predict the onset of diabetes based on diagnostic measures

Data Card Code (2802) Discussion (51)

About Dataset

Context

This dataset is originally from the National Institute of Diabetes and Digestive and Kidney Diseases. The objective of the dataset is to diagnostically predict whether or not a patient has diabetes, based on certain diagnostic measurements included in the dataset. Several constraints were placed on the selection of these instances from a larger database. In particular, all patients here are females at least 21 years old of Pima Indian heritage.

Content

The datasets consists of several medical predictor variables and one target variable, `Outcome`. Predictor variables includes the number of pregnancies the patient has had, their BMI, insulin level, age, and so on.

Acknowledgements

Smith, J.W., Everhart, J.E., Dickson, W.C., Knowler, W.C., & Johannes, R.S. (1988). Using the ADAP learning algorithm to forecast the onset of diabetes mellitus. In *Proceedings of the Symposium on Computer Applications and Medical Care* (pp. 261--265). IEEE Computer Society Press.

Usability 8.82

License CC0: Public Domain

Expected update frequency Not specified

Tags Earth and Nature Health Diabetes India Healthcare

※ 출처 : <https://www.kaggle.com/datasets/uciml/pima-indians-diabetes-database>

□ 데이터 불러오기 및 살펴보기

```
import pandas as pd
```

```
df = pd.read_csv('./diabetes.csv')
```

```
df.head(3)
```

	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

```
df.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

```
df['Outcome'].value_counts()
```

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

데이터는 전부 숫자형이다 → 인코딩 불필요

결과값이 좀 치우쳐져 있다 → Negative가 많으므로 정확도 보다는 재현율 집중

□ Baseline 확인

```
from sklearn.linear_model import LogisticRegression

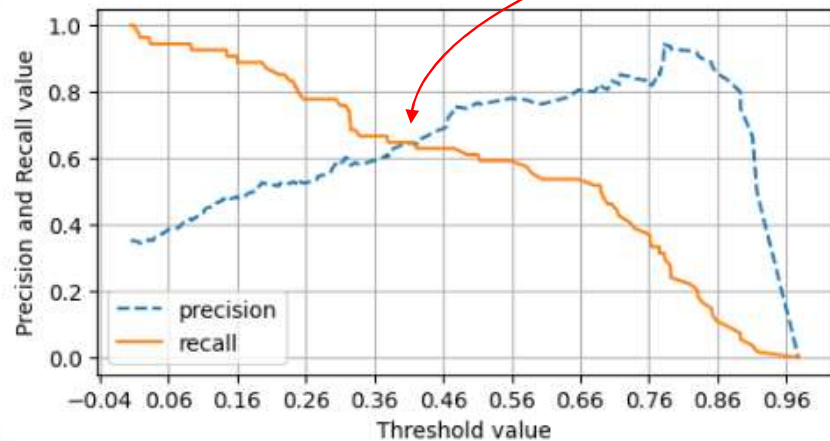
lr_clf = LogisticRegression(max_iter=200)
lr_clf.fit(X_train, y_train)

pred = lr_clf.predict(X_test)
pred_proba_class1 = lr_clf.predict_proba(X_test)[:, 1]

get_clf_eval(y_test, pred, pred_proba_class1)
```

오차 행렬
[[90 10]
 [21 33]]
정확도: 0.7987, 정밀도: 0.7674, 재현율: 0.6111, F1: 0.6804, AUC:0.8072

```
precision_recall_curve_plot(y_test, pred_proba_class1)
```



0.42 정도면 precision과 recall 균형적일 듯
하지만, 0.7이 안되는 수치 → 데이터 정비 필요

□ 결측치 / 0 값 처리

```
df.isnull().sum()
```

```
Pregnancies      0
Glucose           0
BloodPressure     0
SkinThickness     0
Insulin           0
BMI               0
DiabetesPedigreeFunction  0
Age              0
Outcome          0
dtype: int64
```

결측치는 없는데,

0의 값이 너무 많음 → 삭제하기에는 데이터가 부족

```
total_count = df['Glucose'].count()
```

```
zero_features = ['Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI']
```

```
for feature in zero_features:
```

```
    zero_count = df[df[feature] == 0][feature].count()
```

```
    print('{0:>13} 0 건수는 {1:>3}, 퍼센트는 {2:5.2f} %'.format(feature, zero_count, 100*zero_count/total_count))
```

```
      Glucose 0 건수는   5, 퍼센트는   0.65 %
BloodPressure 0 건수는  35, 퍼센트는   4.56 %
SkinThickness 0 건수는 227, 퍼센트는  29.56 %
      Insulin 0 건수는 374, 퍼센트는  48.70 %
         BMI 0 건수는  11, 퍼센트는   1.43 %
```

평균값으로 대체

```
df[zero_features] = df[zero_features].replace(0, df[zero_features].mean())
```

```
for feature in zero_features:
```

```
    zero_count = df[df[feature] == 0][feature].count()
```

```
    print('{0:>13} 0 건수는 {1:>3}, 퍼센트는 {2:5.2f} %'.format(feature, zero_count, 100*zero_count/total_count))
```

```
      Glucose 0 건수는   0, 퍼센트는   0.00 %
BloodPressure 0 건수는   0, 퍼센트는   0.00 %
SkinThickness 0 건수는   0, 퍼센트는   0.00 %
      Insulin 0 건수는   0, 퍼센트는   0.00 %
         BMI 0 건수는   0, 퍼센트는   0.00 %
```

□ Standardization

```
from sklearn.preprocessing import StandardScaler

X = df.iloc[:, :-1]
y = df.iloc[:, -1]

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)

X_train.shape, X_test.shape

((614, 8), (154, 8))

lr_clf = LogisticRegression(max_iter=200)
lr_clf.fit(X_train, y_train)

pred = lr_clf.predict(X_test)
pred_proba_class1 = lr_clf.predict_proba(X_test)[:, 1]

get_clf_eval(y_test, pred, pred_proba_class1)
```

오차 행렬
[[90 10]
[21 33]]
정확도: 0.7987, 정밀도: 0.7674, 재현율: 0.6111, F1: 0.6804, AUC:0.8433

Standardization 적용

살짝 개선

□ Threshold Set

```
from sklearn.preprocessing import Binarizer

def get_eval_by_threshold(y_test, pred_proba_cl, thresholds):

    for custom_threshold in thresholds:
        binarizer = Binarizer(threshold=custom_threshold)
        custom_predict = binarizer.fit_transform(pred_proba_cl.reshape(-1,1))

        print('임계값:', custom_threshold)
        get_clf_eval(y_test, custom_predict, pred_proba_cl)
        print()
```

```
pred_proba_class1 = lr_clf.predict_proba(X_test)[: ,1]
```

```
thresholds = [0.3, 0.33, 0.36, 0.39, 0.42, 0.45, 0.48, 0.50]
get_eval_by_threshold(y_test, pred_proba_class1, thresholds)
```

```
임계값: 0.3
오차 행렬
[[67 33]
 [11 43]]
정확도: 0.7143, 정밀도: 0.5658, 재현율: 0.7963, F1: 0.6615, AUC:0.8433
```

```
임계값: 0.33
오차 행렬
[[72 28]
 [12 42]]
정확도: 0.7403, 정밀도: 0.6000, 재현율: 0.7778, F1: 0.6774, AUC:0.8433
```

```
임계값: 0.36
오차 행렬
[[76 24]
 [15 39]]
정확도: 0.7468, 정밀도: 0.6190, 재현율: 0.7222, F1: 0.6667, AUC:0.8433
```

```
임계값: 0.39
오차 행렬
[[78 22]
 [16 38]]
정확도: 0.7532, 정밀도: 0.6333, 재현율: 0.7037, F1: 0.6667, AUC:0.8433
```

```
임계값: 0.42
오차 행렬
[[84 16]
 [18 36]]
정확도: 0.7792, 정밀도: 0.6923, 재현율: 0.6667, F1: 0.6792, AUC:0.8433
```

```
임계값: 0.45
오차 행렬
[[85 15]
 [18 36]]
정확도: 0.7857, 정밀도: 0.7059, 재현율: 0.6667, F1: 0.6857, AUC:0.8433
```

```
임계값: 0.48
오차 행렬
[[88 12]
 [19 35]]
정확도: 0.7987, 정밀도: 0.7447, 재현율: 0.6481, F1: 0.6931, AUC:0.8433
```

```
임계값: 0.5
오차 행렬
[[90 10]
 [21 33]]
정확도: 0.7987, 정밀도: 0.7674, 재현율: 0.6111, F1: 0.6804, AUC:0.8433
```

```
binarizer = Binarizer(threshold=0.48)
```

```
pred_th_048 = binarizer.fit_transform(pred_proba_class1.reshape(-1,1))
```

```
get_clf_eval(y_test , pred_th_048, pred_proba_class1)
```

```
오차 행렬
[[88 12]
 [19 35]]
정확도: 0.7987, 정밀도: 0.7447, 재현율: 0.6481, F1: 0.6931, AUC:0.8433
```

You've really worked hard today

Next Week ~ ?

04. F1 스코어	166
05. ROC 곡선과 AUC	167
06. 피마 인디언 당뇨병 예측	172
07. 정리	180

04

분류

01. 분류(Classification)의 개요	181
02. 결정 트리	183
결정 트리 모델의 특징	185
결정 트리 파라미터	186
결정 트리 모델의 시각화	187
결정 트리 과적합(Overfitting)	198
결정 트리 실습 - 사용자 행동 인식 데이터 세트	200
03. 앙상블 학습	210
앙상블 학습 개요	210
보팅 유형 - 하드 보팅(Hard Voting)과 소프트 보팅(Soft Voting)	212
보팅 분류기(Voting Classifier)	213
04. 랜덤 포레스트	216
랜덤 포레스트의 개요 및 실습	216
랜덤 포레스트 하이퍼 파라미터 및 튜닝	218
GBM의 개요 및 실습	221
05. GBM(Gradient Boosting Machine)	221
GBM 하이퍼 파라미터 소개	224
XGBoost 개요	225

06. XGBoost(eXtra Gradient Boost)	225
XGBoost 설치하기	227
파이썬 래퍼 XGBoost 하이퍼 파라미터	228
파이썬 래퍼 XGBoost 적용 - 위스콘신 유방암 예측	232
사이킷런 래퍼 XGBoost의 개요 및 적용	240
07. LightGBM	244
LightGBM 설치	246
LightGBM 하이퍼 파라미터	247
하이퍼 파라미터 튜닝 방안	248
파이썬 래퍼 LightGBM과 사이킷런 래퍼 XGBoost,	
LightGBM 하이퍼 파라미터 비교	249
LightGBM 적용 - 위스콘신 유방암 예측	250
08. 베이지안 최적화 기반의 HyperOpt를 이용한	
하이퍼 파라미터 튜닝	253
베이지안 최적화 개요	254
HyperOpt 사용하기	256
HyperOpt를 이용한 XGBoost 하이퍼 파라미터 최적화	262
09. 분류 실습 - 캐글 산탄데르 고객 만족 예측	267
데이터 전처리	268
XGBoost 모델 학습과 하이퍼 파라미터 튜닝	271
LightGBM 모델 학습과 하이퍼 파라미터 튜닝	276
10. 분류 실습 - 캐글 신용카드 사기 검출	279
언더 샘플링과 오버 샘플링의 이해	279
데이터 일차 가공 및 모델 학습/예측/평가	281
데이터 분포도 변환 후 모델 학습/예측/평가	285
이상치 데이터 제거 후 모델 학습/예측/평가	288
SMOTE 오버 샘플링 적용 후 모델 학습/예측/평가	292

Who ~ ?

See you Next Weekend ~ ?