

정수빈/꼬마티탄 / 03.평가

   |  요약 |   공유 Following ...

03.평가



정수빈(이)가 만들

조금 전에 마지막 업데이트 •  조회한 사용자 1명

01. 정확도

02. 오차 행렬

03. 정밀도와 재현율

정밀도/재현율 트레이드오프

정밀도와 재현율의 맹점

정밀도가 100%가 되는 방법

재현율 100%가 되는 방법

04. F1 스코어

05. ROC 곡선과 AUC

06. 피마 인디언 당뇨병 예측


데이터 분석

기본 머신러닝 코드 작성

재현율/정밀도 확인

참고:  머신러닝 평가(Evaluation)

머신러닝은 데이터 가공/변환, 모델 학습/예측, 그리고 평가의 프로세스로 구성됩니다. 앞 장의 타이타닉 생존자 예제에서는 모델 예측 성능의 평가를 위해 정확도를 이용했습니다. 머신러닝 모델은 여러 가지 방법으로 예측 성능을 평가할 수 있습니다. 성능 평가 지표는 일반적으로 모델이 분류나 회귀냐에 따라 여러 종류로 나뉩니다.

 회귀(regression) 는 예측하고자 하는 타겟값이 실수, 즉 숫자인 경우이다. 그리고 회귀는 예측 결과가 연속성을 지닌다.

분류(classification) 는 예측하고자 하는 타겟값이 범주형 변수인 경우이다. 회귀와는 다르게 분류는 예측 결과가 연속성을 지니지 않는다.

회귀의 경우, 실제 값과 예측값의 오차의 평균값에 기반하여 평가합니다.

- 오차의 절대값에 평균오차를 구하는 방법
- 오차의 제곱에 루트를 씌운 뒤 평균 오차를 구하는 방법

분류의 경우

- 정확도 (Accuracy)
- 오차 행렬
- 정밀도 (Precision)
- 재현율 (Recall)
- F1스코어

- ROC AUC

이번 장에서는 부류의 평가에 대해서 배울 예정입니다.

정수빈/꼬마티탄 / 03.평가



01. 정확도

정확도는 실제 데이터에서 예측 데이터가 얼마나 같은지를 판단하는 지표입니다. 정확도는 직관적으로 모델 예측 성능을 나타내는 평가 지표입니다.



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

이진 분류의 경우 데이터의 구성에 따라 ML모델의 성능을 왜곡할 수 있기 때문에 정확도 수치 하나만 가지고 성능을 평가하지 않습니다. 그럼 정확도 수치가 어떻게 모델의 성능을 왜곡 시킬 수 있을까요?

이전 장의 타이타닉 예제에서 ML알고리즘을 사용했을 때 예측이 85%정도 정확했습니다. 데이터를 보면, 남자보다 여자가 생존확률이 높았습니다. 그렇다면 여자면 무조건 생존, 남자는 사망으로 예측한다면 어떻게 될까요? 정확도는 0.8324 로 상당히 높은 수치가 나왔습니다. (책에서는 78%가 나왔다고 합니다.)

› MyDummyClassifier 구현

이렇기에 **정확도**를 평가 지표로 사용할 때는 신중해야 합니다. 예를 들어 100개의 데이터가 있고 이 중 90개의 데이터 레이블이 0, 단 10개의 데이터 레이블이 1일때, 무조건 0으로 예측해도 90%의 정확도가 나올 수 있습니다.

02. 오차 행렬

이진 분류에서 성능 지표로 잘 활용되는 오차행렬(confusion matrix)은 학습된 분류 모델이 예측을 수행하면서 얼마나 헷갈리고 (confused) 있는지도 함께 보여주는 지표입니다. 즉, 이진 분류의 예측 오류가 얼마인지 더불어 어떠한 유형의 예측 오류가 발생하고 있는지를 함께 나타내는 지표입니다.

```
1 from sklearn.metrics import confusion_matrix
2
3 confusion_matrix(y_test, dm_pred)
4 --
5 [[103  15]
6  [ 15  46]]
```

예측 클래스
(Predicted Class)

Negative(0)

P

tive(0)

TN
(True Negative)

(False)

정수빈/꼬마티탄 / 03.평가



tive(1)

FN
(False Negative)

(True)

- TN: 예측값을 Negative 값 0으로 예측했고 실제 값 역시 Negative 값 0
- FP: 예측값을 Positive 값 1로 예측했는데 실제 값은 Negative 값 0
- FN: 예측값을 Negative 값 0으로 예측했는데 실제 값은 Positive 값 1
- TP: 예측값을 Positive 값 1로 예측했고 실제 값 역시 Positive 값 1



정확도 = 예측 결과와 실제 값이 동일한 건수 / 전체 데이터수 = $(TN + TP) / (TN + FP + FN + TP)$

불균형한 이진 분류 데이터 세트에서는 Positive 데이터 건수가 매우 작기 때문에 데이터에 기반한 ML알고리즘은 Positive보다는 Negative로 예측 정확도가 높아지는 경향이 있습니다. 10,000건의 데이터 세트에서 9,900건이 Negative고 100건만이 Positive이면 Positive에 대한 정확도는 파악하지 못하고 Negative에 대한 정확도만 높아지게 됩니다.

정확도는 분류(Classification)모델의 성능을 측정할 수 있는 한 가지 요소일 뿐입니다. 불균형한 데이터 세트에서 정확도만으로는 모델 신뢰도가 떨어질 수 있습니다.

03. 정밀도와 재현율

정밀도와 재현율은 Positive 데이터 세트의 예측 성능에 좀 더 초점을 맞춘 평가 지표입니다. 항상 Negative로만 분류하는 분류기가 있다고 해봅시다. 그렇다면 정밀도와 재현율 전부 0입니다.



정밀도 = $TP / (FP + TP)$ = 예측을 Positive으로 한 대상 중 실제 값이 Positive으로 일치한 데이터의 비율

재현율 = $TP / (FN + TP)$ = 실제 값이 Positive인 대상 중에 예측과 실제 값이 Positive으로 일치한 데이터의 비율

재현율은 민감도 또는 TPR(True Positive Rate)라고도 불립니다. 재현율이 중요 지표인 경우는 실제 Positive 양성 데이터를 Negative로 잘못 판단하게 되면 업무상 큰 영향이 발생하는 경우입니다. 예를 들자면, 암 판단 모델, 금융 사기 적발 모델이 있습니다.

보통 재현율이 정밀도보다 상대적으로 중요한 업무가 많지만, 정밀도가 더 중요한 경우가 있습니다. 정밀도는 실제 Negative 음성인 데이터 예측을 Positive 양성으로 잘못 판단하게 되면 업무상 큰 영향을 발생하는 경우입니다.

다. 예를 들어, 스팸메일 여부입니다.

정수빈/꼬마티탄 / 03.평가



```

2 from sklearn.metrics import precision_score, recall_score, confusion_matrix,
3
4 def get_clf_eval(y_test, pred):
5     confusion = confusion_matrix(y_test, pred)
6     accuracy = accuracy_score(y_test, pred)
7     precision = precision_score(y_test, pred)
8     recall = recall_score(y_test, pred)
9     print('오차 행렬')
10    print(confusion)
11    print("정확도 : {0:.4f}, 정밀도 : {1:.4f}, 재현율 : {2:.4f}".format(accuracy,
12
13    ### 데이터 로드 부분 생략 ###
14
15    lr_clf = LogisticRegression(solver='liblinear')
16
17    lr_clf.fit(X_train, y_train)
18    lr_pred = lr_clf.predict(X_test)
19
20    get_clf_eval(y_test, lr_pred)

```

결과

```

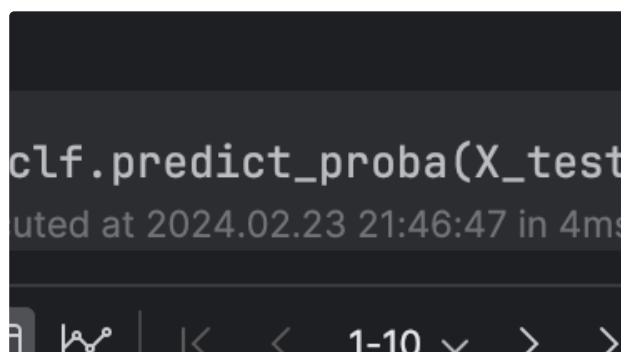
1 오차 행렬
2 [[108  10]
3  [ 14  47]]
4 정확도 : 0.8659, 정밀도 : 0.8246, 재현율 : 0.7705

```

정밀도/재현율 트레이드오프

분류하려는 업무의 특성상 정밀도 또는 재현율이 특별히 강조돼야 할 경우 분류의 결정 임계값을 조정해 정밀도 또는 재현율의 수치를 높일 수 있습니다. 하지만 정밀도와 재현율은 상호 보완적이기 때문에 어느 한 쪽을 강제로 높이면 다른 한의 수치는 떨어지기 쉽습니다. 이를 정밀/재현율 트레이드오프라고 부릅니다.

이전 장의 타이타닉 생존 확률 예측 모델을 생각해봅시다. 첫번째 테스트 데이터에서 승객이 사망할 확률이 45%고 생존할 확률이 55%라고 예측이라고 한다면, 생존할 확률이 더 크기 때문에 생존으로 예측을 합니다. 일반적인 이진 분류에서는 이 임계값을 0.5로 정하고 이 기준 값보다 크면 Positive, 작으면 Negative로 결정합니다.



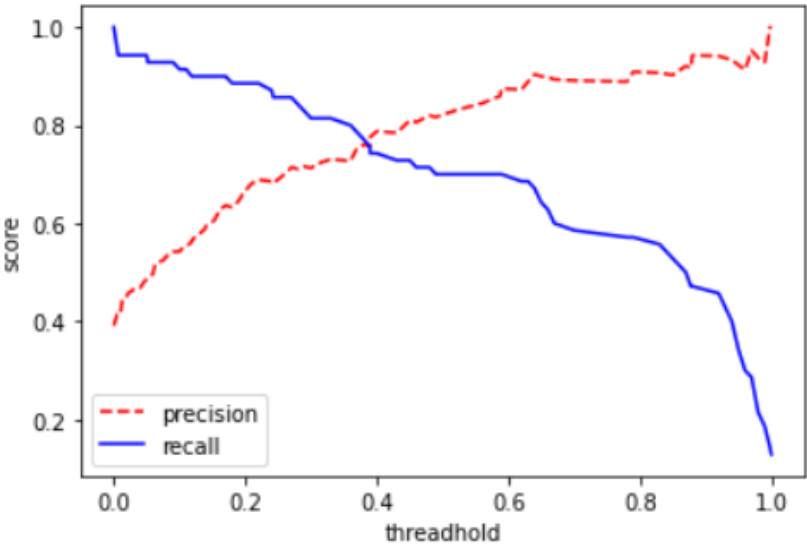
정수빈/꼬마티탄 / 03.평가



1	0.863752	0.1362
2	0.861586	0.1384
3	0.846385	0.1536
4	0.823489	0.1765
5	0.842403	0.1575
6	0.871336	0.1286
7	0.274057	0.7259

타이타닉 테스트 데이터마다 사망(0), 생존(1) 확률표

하지만 임계값이 0.5인 것이 가장 믿을 만한 평가를 내지 않습니다. 아래는 임계값을 변경하면서 정밀도, 재현도의 점수를 그래프로 나타낸 것입니다.



재현도/정밀도 트레이드오프 예시그림

정밀도와 재현율의 맹점

Positive 임계값은 변경함에 따라 정밀도와 재현율의 수치가 변경됩니다. 임계값의 변경은 업무 환경에 맞게 두 개의 수치를 상호 보완할 수 있을 수준에서 적용돼야 합니다.

정밀도가 100%가 되는 방법

확실한 기준이 되는 경우만 Positive 예측하고 나머지는 모두 Negative로 예측합니다. 예를 들어, 환자가 80세

정수빈/꼬마티탄 / 03.평가



이 한명만 Positive 나머지는 전부 Negative로 예측한다면 정밀도는 $TP / (TP + FP)$ 이므로 $1 / (1 + 0) = 1 = 100\%$ 가 됩니다.

재현율 100%가 되는 방법

모든 환자를 Positive로 예측하면 됩니다. 재현율은 $TP / (TP + FN)$ 이므로 실제 양성환자가 30명이라고 했을 때 $30 / (30 + 0) = 100\%$ 이 됩니다.

이처럼 정밀도와 재현율 성능 수치 중 어느 한 쪽만 참조하면 극단적으로 수치 조작이 가능해집니다.

04. F1 스코어

F1 스코어는 정밀도와 재현율을 결합한 지표입니다. 정밀도가 0.9, 재현율 0.1으로 극단적으로 차이나는 모델보다 정밀도와 재현율이 0.5로 비슷한 모델이 있다면 F1 스코어는 후자의 모델이 더 좋은 모델이라고 판단합니다.

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

```
1 from sklearn.metrics import f1_score
2 f1 = f1_score(y_test, lr_pred)
3 print('F1 스코어 : {0:.4f}'.format(f1))
4
5 # F1 스코어 : 0.7966
```

05. ROC 곡선과 AUC

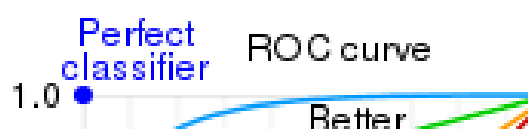
ROC 곡선과 이에 기반한 AUC 스코어는 이진 분류의 예측 성능 측정에서 중요하게 사용되는 지표입니다.

ROC(Receiver Operation Characteristic Curve) 곡선은 수신자 판단 곡선으로 불립니다. ROC 곡선은 FPR(False Positive Rate)이 변할 때 TPR(True Positive Rate)이 어떻게 변하는지 나타내는 곡선입니다.

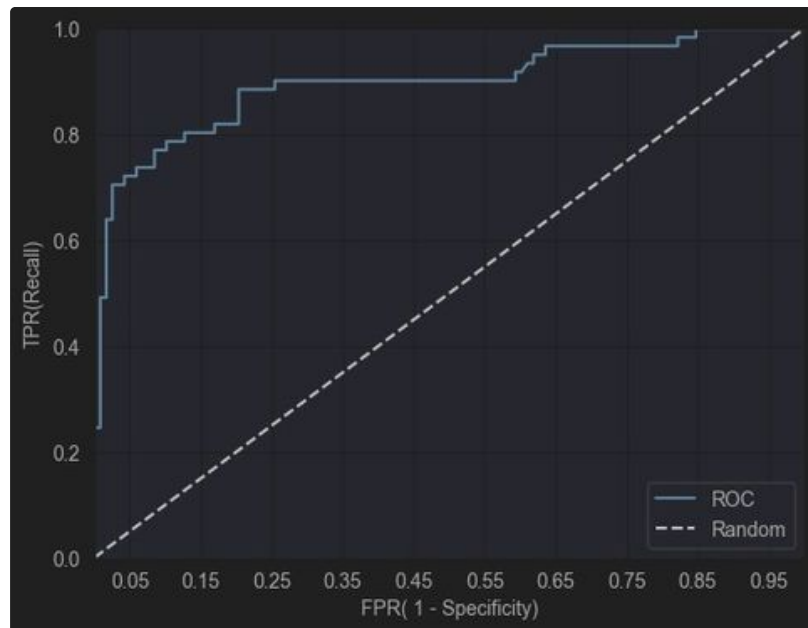
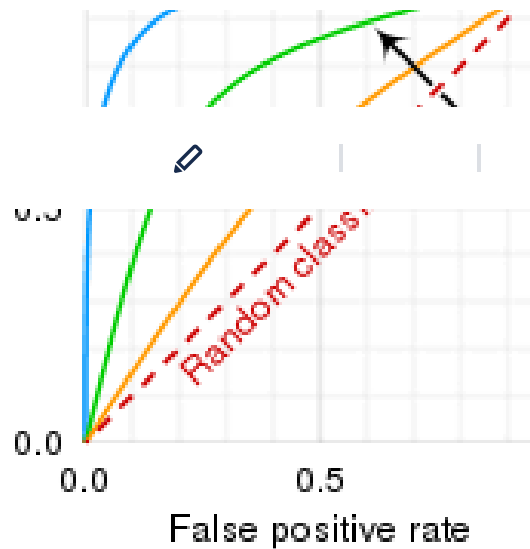
TNR(True Negative Rate)은 특이성은 실제 Negative가 정확히 예측돼야하는 수준을 나타냅니다. 실제로 질병이 없는 건강한 사람은 질병이 없는 것으로 음성판단하는 것을 말합니다.

$$\text{Rainbow} \quad TNR = TN / (TN + FP)$$

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



정수빈/꼬마티탄 / 03.평가



타이타닉 ROC 곡선

AUC(Area Under Curve) 값은 ROC 곡선 밑의 면적을 구한 것으로서 일반적으로 1에 가까울수록 좋은 수치입니다. 가운데 직선에서 멀어지고 직사각형에 가까운 곡선이 되어 면적이 1에 가까워지게 되면 좋은 ROC AUC 성능 수치를 얻게 됩니다. 가운데 곡선은 동전던지기 수준의 랜덤입니다.

06. 피마 인디언 당뇨병 예측

이번에는 피마 인디언 당뇨병 데이터 세트를 이용해 당뇨병 여부를 판단하는 머신 러닝 예측 모델을 수립하고, 지금까지 설명한 평가 지표를 적용해 보겠습니다.

데이터 분석

```
1 import numpy as np
2 import pandas as pd
3
4 diabetes_data = pd.read_csv('diabetes.csv')
5 print(diabetes_data.head())
```

123 Pregnancies	123 Glucose	123 BloodPressure	123 SkinThickness	123 Insulin	123 BMI	123 Diabete...	123 Ag
0	6	148.0	72.0	35.000000	79.799479	33.6	0.627

정수빈/꼬마티탄 / 03.평가



4	0	137.0	40.0	35.000000	168.000000	43.1	2.288
---	---	-------	------	-----------	------------	------	-------

```
diabetes_data.head()
```

데이터의 Outcome 컬럼이 당뇨병 발병 유무를 체크하는 컬럼입니다.

```
1 diabetes_data.info()
```

#	Column	Non-Null Count	Dtype
---	-----	-----	-----
0	Pregnancies	768 non-null	int64
1	Glucose	768 non-null	float64
2	BloodPressure	768 non-null	float64
3	SkinThickness	768 non-null	float64
4	Insulin	768 non-null	float64
5	BMI	768 non-null	float64
6	DiabetesPedigreeFunction	768 non-null	float64
7	Age	768 non-null	int64
8	Outcome	768 non-null	int64
dtypes: float64(6), int64(3)			
memory usage: 54.1 KB			

이번에는 타이타닉 컬럼과 다르게 결측값이 존재하지 않고, 전부 숫자형태이므로 따로 변형할 필요는 없습니다.

```
1 diabetes_data.describe()
```

123 Pregnancies	123 Glucose	123 BloodPressure	123 SkinThickness	123 Insulin	123 BMI	123 Di
count	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
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000

기본 머신러닝 코드 작성

```
1 import numpy as np
2 import pandas as pd
3 from sklearn.linear_model import LogisticRegression
4 from sklearn.model_selection import train_test_split
5
```



```
6 diabetes_data = pd.read_csv('diabetes.csv')
```

```
7
```

정수빈/꼬마티탄 / 03.평가

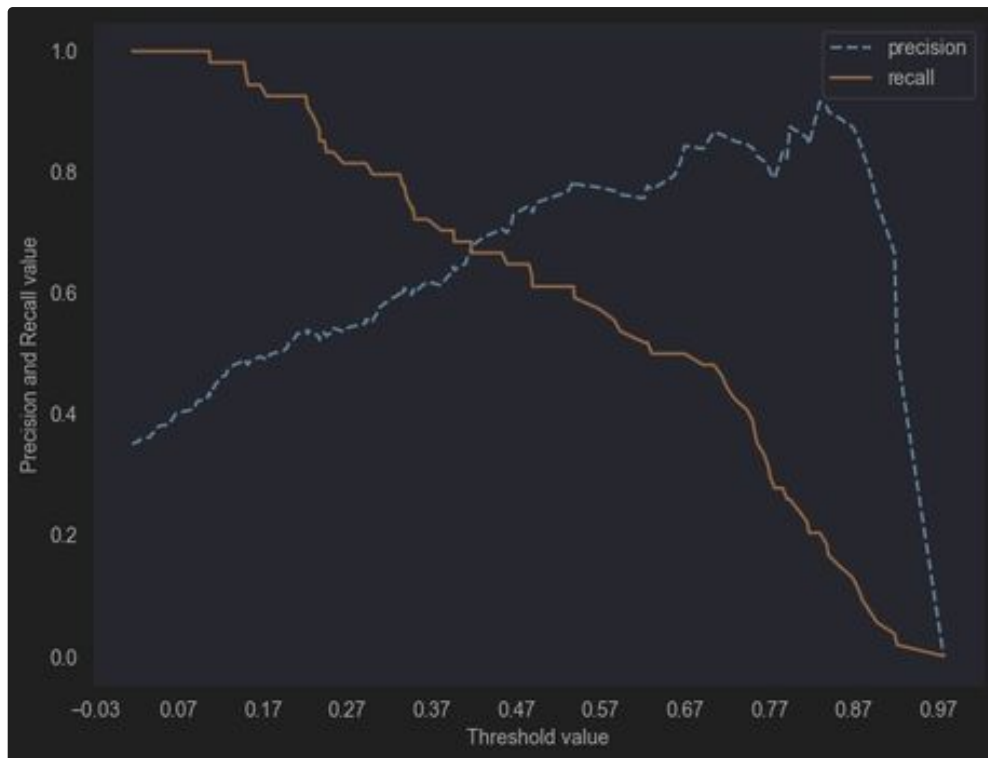


```
11 X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, rand
12
13 lr_clf = LogisticRegression(solver='liblinear')
14 lr_clf.fit(X_train, Y_train)
15 pred = lr_clf.predict(X_test)
```

재현율/정밀도 확인

› get_clf_eval 함수 / precision_recall_curve_plot 함수

```
1 pred_proba_c1 = lr_clf.predict_proba(X_test)[: , 1]
2 precision_recall_curve_plot(Y_test, pred_proba_c1)
```



```
1 get_clf_eval(Y_test, pred, lr_clf.predict_proba(X_test)[: , 1])
2 정확도 : 0.7987, 정밀도 : 0.7674, 재현율 : 0.6111 F1 : 0.6804, AUC : 0.8433
```

재현율 곡선을 보면 임계값을 0.42 정도로 낮추면 정밀도와 재현율이 어느정도 균형을 맞출 것 같습니다. 하지만 두 개의 지표 모두 0.7이 안되는 수치로 보입니다. 여전히 두 지표의 값이 낮습니다. 임계값을 인위적으로 조작하기 전에 다시 데이터 값을 점검하겠습니다. 먼저 원본 데이터를 다시 살펴봅시다.

```
1 diabetes_data.describe()
```

	123 Pregnancies	123 Glucose	123 BloodPressure	123 SkinThickness	123 Insulin	123 BMI
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	121.681605	72.254807	26.606479	118.660163	32.450805
std	3.369578	30.436016	12.115932	9.631241	93.080358	6.875374
min	0.000000	44.000000	24.000000	7.000000	14.000000	18.200000
25%	1.000000	99.750000	64.000000	20.536458	79.799479	27.500000
50%	3.000000	117.000000	72.000000	23.000000	79.799479	32.000000
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000

정수빈/꼬마티탄 / 03.평가



Glucose, BloodPressure, SkinThickness, Insulin, BMI 의 min값이 0인 것을 알 수 있습니다. 0 값의 건수 및 전체 데이터 건수 대비 몇 퍼센트 존재하지는 확인하면 아래와 같습니다.

- Glucose 0 건수는 5, 퍼센트는 0.651042 %
- BloodPressure 0 건수는 35, 퍼센트는 4.557292 %
- SkinThickness 0 건수는 227, 퍼센트는 29.557292 %
- Insulin 0 건수는 374, 퍼센트는 48.697917 %
- BMI 0 건수는 11, 퍼센트는 1.432292 %

SkinThickness, Insulin의 0값은 각각 전체의 약 30%, 50%이기 때문에 일괄적으로 삭제한다면 학습을 효과적으로 수행할 수 없기 때문에 0을 평균값으로 대체하겠습니다.

```
1 mean_zero_features = diabetes_data[zero_features].mean()
2 diabetes_data[zero_features] = diabetes_data[zero_features].replace(0, mean_ze
```

	123 Pregnancies	123 Glucose	123 BloodPressure	123 SkinThickness	123 Insulin	123 BMI
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	121.681605	72.254807	26.606479	118.660163	32.450805
std	3.369578	30.436016	12.115932	9.631241	93.080358	6.875374
min	0.000000	44.000000	24.000000	7.000000	14.000000	18.200000
25%	1.000000	99.750000	64.000000	20.536458	79.799479	27.500000
50%	3.000000	117.000000	72.000000	23.000000	79.799479	32.000000
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000

변형 후

```
1 from sklearn.preprocessing import StandardScaler
2
3 X = diabetes_data.iloc[:, :-1]
4 Y = diabetes_data.iloc[:, -1]
5
6 scaler = StandardScaler()
7 X_scaled = scaler.fit_transform(X)
8
9 X_train, X_test, Y_train, Y_test = train_test_split(X_scaled, Y, test_size=0.
10
11 lr_clf = LogisticRegression(solver='liblinear')
```

```

12 lr_clf.fit(X_train, Y_train)
13 pred = lr_clf.predict(X_test)

```

정수빈/꼬마티탄 / 03.평가



LogisticRegression에서는 StandardScaler를 해주는 게 결과에 더 좋다고 합니다.

› get_eval_by_threshold 함수

```

1 thresholds = [0.3, 0.33, 0.36, 0.39, 0.42, 0.45, 0.48, 0.50]
2 pred_proba = lr_clf.predict_proba(X_test)
3 get_eval_by_threshold(Y_test, pred_proba[:, 1].reshape(-1, 1), thresholds)

```

```

임계값 : 0.3
정확도 : 0.7013, 정밀도 : 0.5513, 재현율 : 0.7963 F1 : 0.6515, AUC : 0.8433
임계값 : 0.33
정확도 : 0.7403, 정밀도 : 0.5972, 재현율 : 0.7963 F1 : 0.6825, AUC : 0.8433
임계값 : 0.36
정확도 : 0.7468, 정밀도 : 0.6190, 재현율 : 0.7222 F1 : 0.6667, AUC : 0.8433
임계값 : 0.39
정확도 : 0.7532, 정밀도 : 0.6333, 재현율 : 0.7037 F1 : 0.6667, AUC : 0.8433
임계값 : 0.42
정확도 : 0.7792, 정밀도 : 0.6923, 재현율 : 0.6667 F1 : 0.6792, AUC : 0.8433
임계값 : 0.45
정확도 : 0.7857, 정밀도 : 0.7059, 재현율 : 0.6667 F1 : 0.6857, AUC : 0.8433
임계값 : 0.48

```

```

1 binarizer = Binarizer(threshold=0.48)
2 pred_th_048 = binarizer.fit_transform(pred_proba[:, 1].reshape(-1, 1))
3
4 get_clf_eval(Y_test, pred_th_048, pred_proba[:, 1])
5
6 정확도 : 0.7987, 정밀도 : 0.7447, 재현율 : 0.6481 F1 : 0.6931, AUC : 0.8433

```

+ 레이블 추가

첫 번째로 반응을 추가해 보세요