

## 평가 지표 및 측정

### 1.1.1 최종 목표를 기억하라

### 1.1.2 이진 분류의 평가지표

### 1.1.3 다중 분류의 평가지표

### 1.1.4 회귀의 평가 지표

## 학습 내용

- 이진 분류의 평가 지표에 대해 알아본다.
- 불균형 데이터 셋일때의 정확도에 대해 알아본다.
- 정밀도, 민감도, 특이도, FPRate, F-score에 대해 알아본다.
- 함수를 활용하여 각각의 모델별 정밀도, 민감도, F-score를 확인해 본다.

In [1]:

```
from IPython.display import display, Image
```

In [2]:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import seaborn as sns
```

## 01 데이터 셋 준비

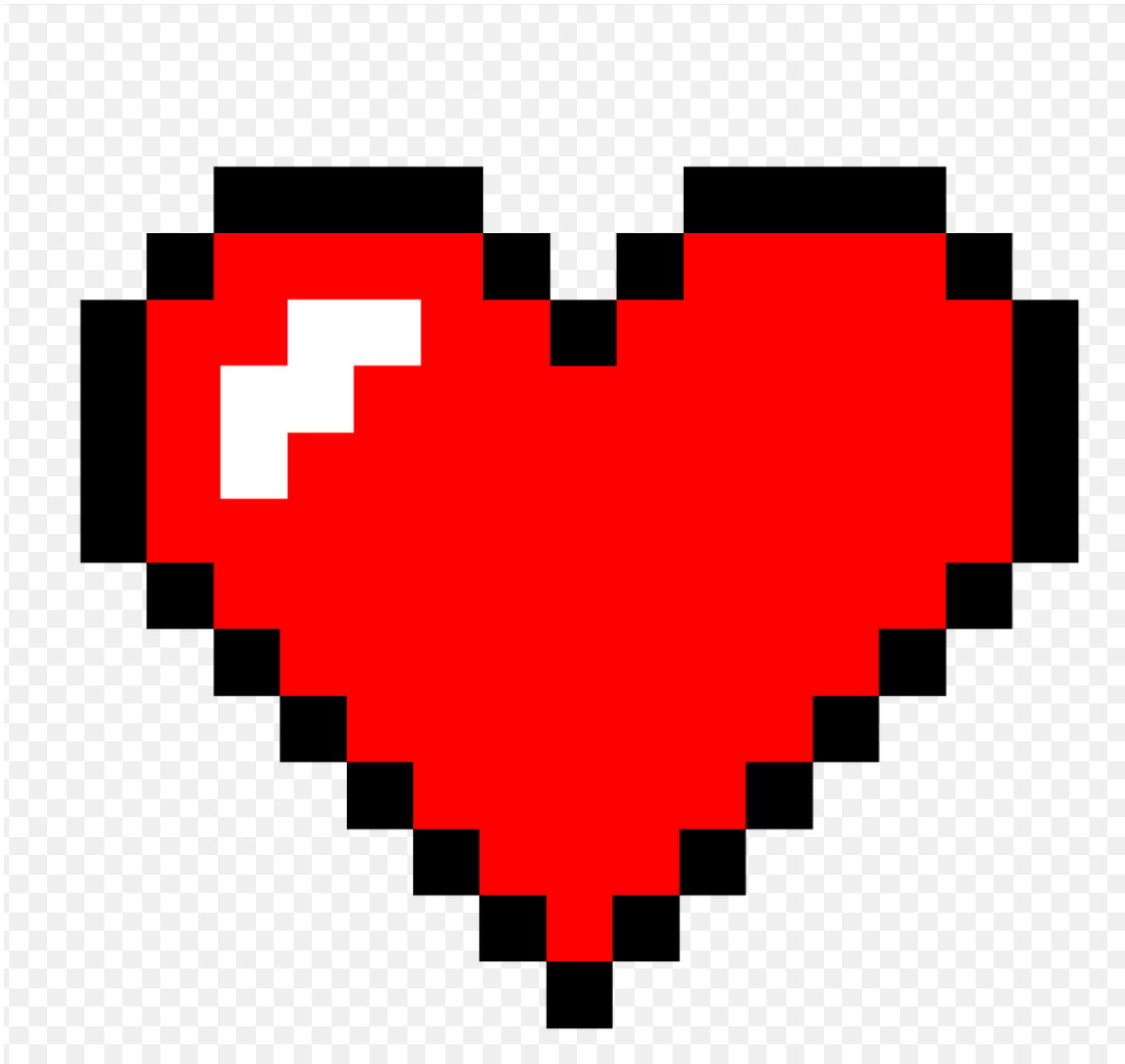
### 데이터 셋

- 손글씨 데이터
- data : 1797장, 64개의 pixel 데이터
  - images : 1797, 8, 8
- target : 0~9까지의 손글씨 값
- pixel : 화소(텔레비전·컴퓨터 화면의 화상을 구성하는 최소 단위)
  - 화면 이미지들은 더 이상 쪼개지지 않는 사각형의 작은 점들이 모여 이뤄진다. 이때 이미지를 구성하는 최소 단위를 픽셀이라고 한다.

In [4]:



```
## 머신러닝 작업 flow  
display(Image(filename='img/model_validation_pixel01.png'))
```



In [5]:



```
from sklearn.datasets import load_digits

digits = load_digits()
print(digits.data.shape)
print(digits.keys(), digits.target)
print(np.unique( digits.target ) )
sns.countplot(digits.target)
```

(1797, 64)

```
dict_keys(['data', 'target', 'frame', 'feature_names', 'target_names', 'images', 'DESCR']) [0 1 2 ... 8 9 8]
```

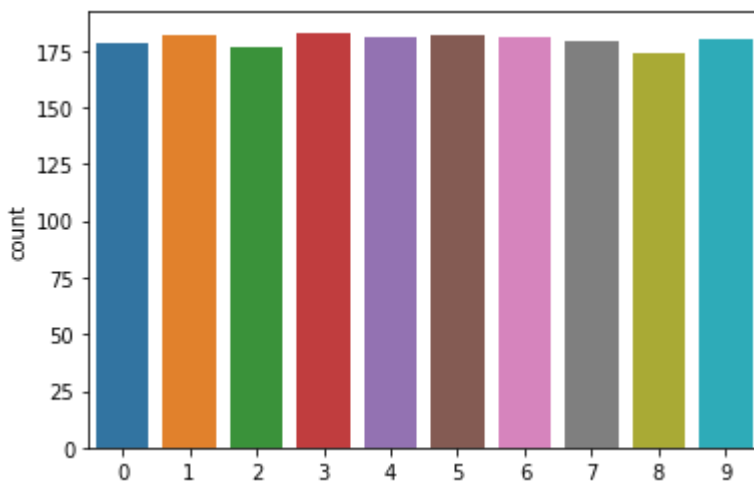
```
[0 1 2 3 4 5 6 7 8 9]
```

C:\Users\Wtoto\Anaconda3\lib\site-packages\seaborn\decorators.py:36: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.

```
warnings.warn(
```

Out[5]:

```
<AxesSubplot:ylabel='count'>
```



## 데이터 셋의 Target(타겟)이 9:1의 비율로 나누기

- 9이면 True
- 9가 아니면 False

In [6]:



```
X = digits.data          # 입력
y = digits.target == 9    # 출력

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
```

In [8]:



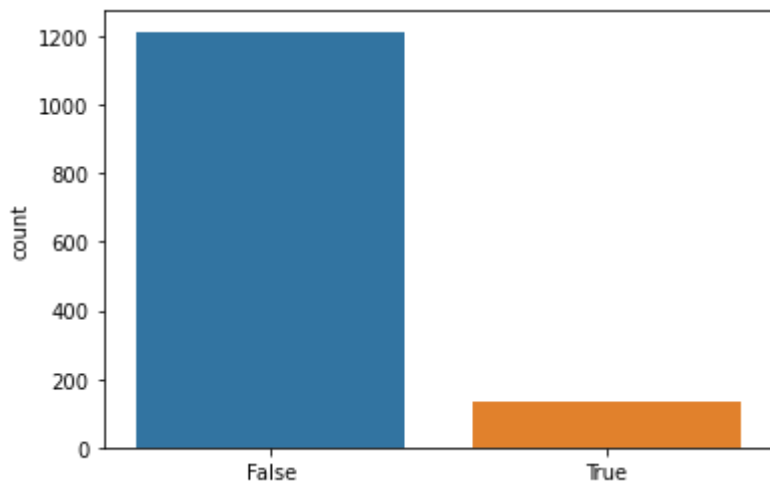
```
# y_train의 값 확인  
sns.countplot(y_train)
```

C:\Users\Wtoto\Anaconda3\lib\site-packages\seaborn\decorators.py:36: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.

```
warnings.warn(
```

Out[8]:

<AxesSubplot:ylabel='count'>



In [9]:

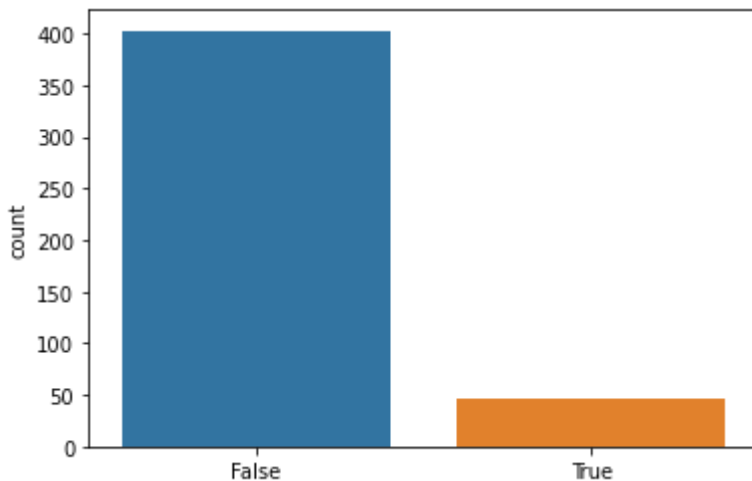
```
# y_test의 값 확인  
sns.countplot(y_test)
```

C:\Users\Wtoto\Anaconda3\lib\site-packages\seaborn\decorators.py:36: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.

```
warnings.warn(
```

Out[9]:

<AxesSubplot:ylabel='count'>



## 02 다양한 모델의 평가

- 정확도(accuracy) 확인

### 02-01 기본 모델 DummyClassifier

- 간단한 규칙을 사용하여 예측을 수행한다.
- 실제 프로젝트에서 사용하지 않으며, 간단한 베이스라인 모델로서 사용된다.
- DummyClassifier(strategy='most\_frequent') : 학습용 세트에서 가장 많이 있는 Label(라벨)을 예측한다.
  - most\_frequent : 가장 많이 있는 Label(라벨)을 예측
  - stratified : 클래스 분포를 존중하여 예측을 생성
  - uniform : 무작위로 균일하게 예측을 생성 ,
  - 기타 : prior, constant
- 아래 모델은 가장 많은 레이블을 가진 False만 예측하게 된다.

In [14]:



```
from sklearn.dummy import DummyClassifier
dummy_model = DummyClassifier(strategy='most_frequent').fit(X_train, y_train)
pred_most_frequent = dummy_model.predict(X_test)
print("예측된 레이블의 고유값: {}".format(np.unique(pred_most_frequent)))
print("테스트 점수: {:.2f}".format(dummy_model.score(X_test, y_test)))
```

예측된 레이블의 고유값: [False]

테스트 점수: 0.90

## 02-02 DummyClassifier를 이용한 예측

- 매개변수 없을 때의 기본 동작
  - stratified : 클래스 분포를 고려하여 예측
- 클래스의 9:1 분포를 가만하여 예측

In [15]:



```
dummy = DummyClassifier(strategy='stratified').fit(X_train, y_train)
pred_dummy = dummy.predict(X_test)
print("예측된 레이블의 고유값: {}".format(np.unique(pred_dummy)))
print("dummy 점수: {:.2f}".format(dummy.score(X_test, y_test)))
```

예측된 레이블의 고유값: [False True]

dummy 점수: 0.82

## 02-03 실제 모델을 사용 - DecisionTreeClassifier

In [16]:



```
from sklearn.tree import DecisionTreeClassifier
tree = DecisionTreeClassifier(max_depth=2).fit(X_train, y_train)
pred_tree = tree.predict(X_test)
print("테스트 점수: {:.2f}".format(tree.score(X_test, y_test)))
```

테스트 점수: 0.92

- 실제 분류기와 기본 모델 dummy 분류기와 성능차이가 거의 없다.

## 02-04 LogisticRegression 모델 확인

In [17]:

```
from sklearn.linear_model import LogisticRegression

logreg = LogisticRegression(C=0.1).fit(X_train, y_train)
pred_logreg = logreg.predict(X_test)
print("logreg 점수: {:.2f}".format(logreg.score(X_test, y_test)))
```

logreg 점수: 0.98

C:\Users\Wtoto\Anaconda3\lib\site-packages\sklearn\linear\_model\\_logistic.py:762: ConvergenceWarning: lbfgs failed to converge (status=1):  
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max\_iter) or scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html> (<https://scikit-learn.org/stable/modules/preprocessing.html>)

Please also refer to the documentation for alternative solver options:

[https://scikit-learn.org/stable/modules/linear\\_model.html#logistic-regression](https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression) ([https://scikit-learn.org/stable/modules/linear\\_model.html#logistic-regression](https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression))

```
n_iter_i = _check_optimize_result(
```

**하나만 예측하는 기본 모델도 90% 이상의 정확도를 갖는다.**

- 정확도는 때로는 평가지표로 사용하기에 부족한 부분이 있다.

**정확도 대신에 사용할 지표가 무엇이 있을까?**

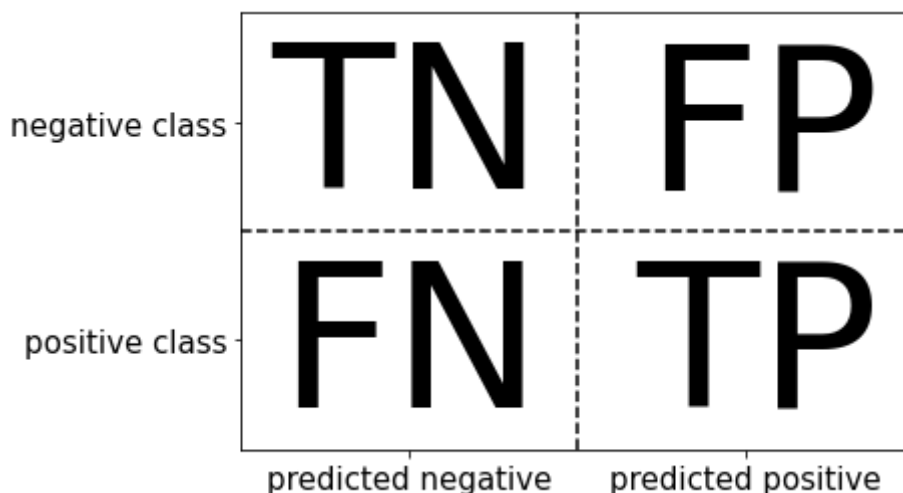
### 03 오차행렬(confusion matrix)을 이용하기

In [18]:

```
import mglearn
```

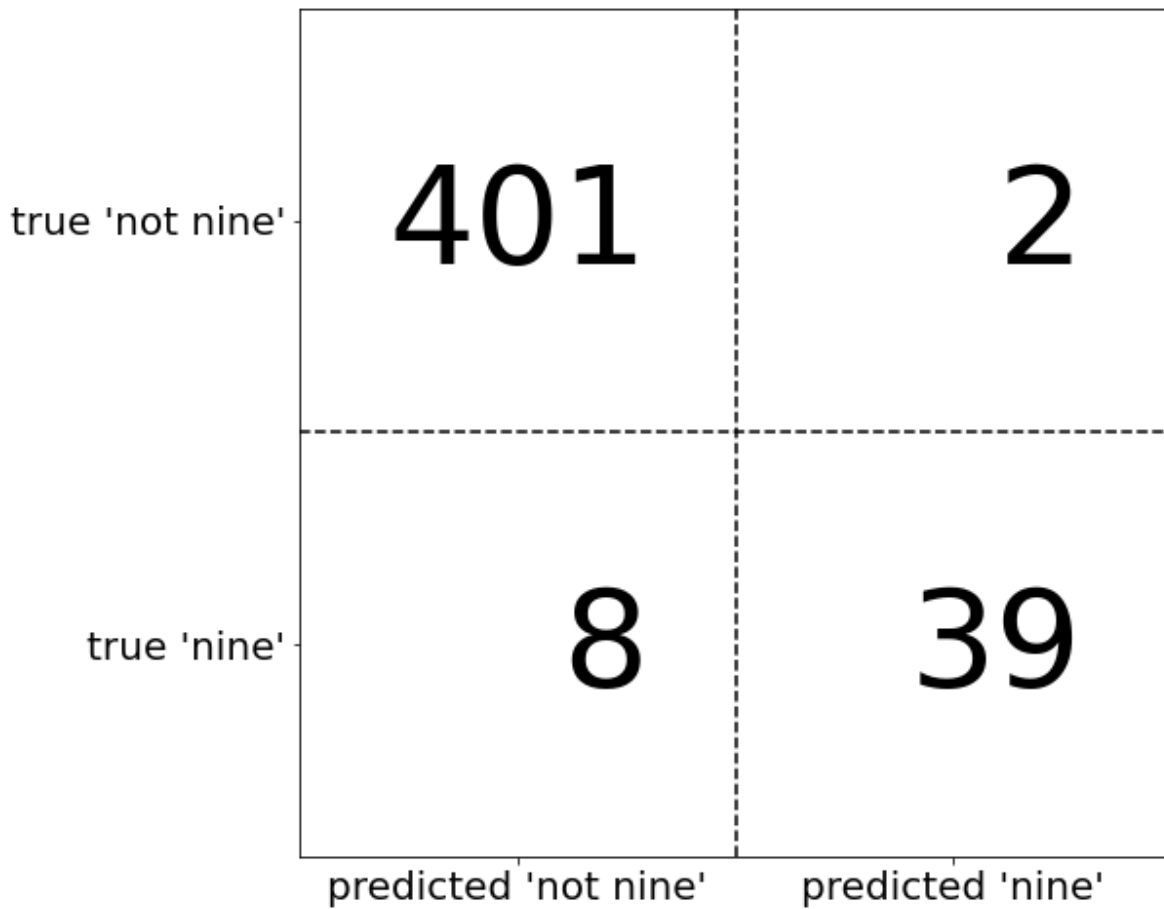
In [19]:

```
mglearn.plots.plot_binary_confusion_matrix()
```



In [20]:

```
mglearn.plots.plot_confusion_matrix_illustration()
```



### confusion\_matrix 를 이용한 오차 행렬 구하기

In [21]:

```
from sklearn.metrics import confusion_matrix

confusion = confusion_matrix(y_test, pred_logreg)
print("오차 행렬:Wn{}".format(confusion))
```

오차 행렬:  
[[402 1]  
[ 6 41]]

### 3-1 각각의 예측값에 대한 오차행렬을 확인해보기



In [22]:



```
print("빈도 기반 더미 모델:")
print(confusion_matrix(y_test, pred_most_frequent))

print("\n무작위 더미 모델:")
print(confusion_matrix(y_test, pred_dummy))

print("\n결정 트리:")
print(confusion_matrix(y_test, pred_tree))

print("\n로지스틱 회귀")
print(confusion_matrix(y_test, pred_logreg))
```

빈도 기반 더미 모델:

```
[[403  0]
 [ 47  0]]
```

무작위 더미 모델:

```
[[374 29]
 [ 47  0]]
```

결정 트리:

```
[[390 13]
 [ 24 23]]
```

로지스틱 회귀

```
[[402  1]
 [  6 41]]
```

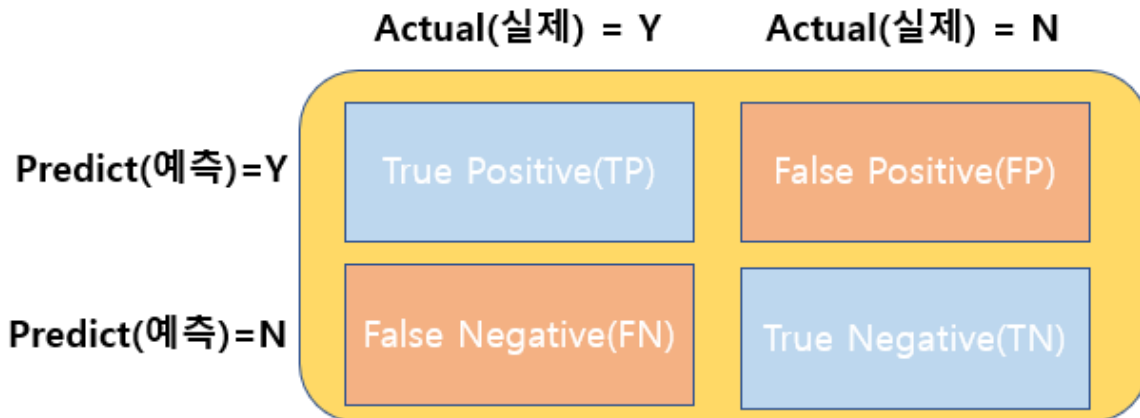
## 3-2 분류의 다양한 평가지표를 살펴보기

- 정확도(accuracy) : 정확하게 예측/전체 예측수
- 정밀도(precision) : 양성으로 예측한 것중(TP+FP), 진짜 양성(TP)
- 민감도(sensitivity), 재현율(recall) : 전체 양성 샘플(TP + FN)중에서 얼마나 많은 샘플이 양성 클래스로 분류 (TP)
- 특이도
- Fprate
- F-score
- AUC

In [23]:



```
## 머신러닝 작업 flow
display(Image(filename='img/model_validation01.png'))
```



분류의 평가지표를 살펴보자.

**정확도(accuracy) : 정확하게 예측/전체 예측수**

$$\text{accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{FP} + \text{FN} + \text{TN}}$$

**정밀도(precision) : 예측을 양성(Positive)으로 한 것 전체(TP+FP)중에 잘 예측한 것 (TP)**

$$\text{정밀도(precision)} = \frac{\text{잘 예측(TP)}}{\text{예측을 양성으로 한 것 전체(TP+FP)}}$$

- 언제 사용하는가?: 거짓 양성(FP)의 수를 줄일 때 사용
  - 임상 실험을 통해 신약의 치료 효과를 예측하는 모델
  - 임상 실험은 가격이 매우 비싸, 제약회사는 한번의 임상실험으로 신약의 효과를 검증하기를 원함. 모델이 거짓 양성(FP)을 많이 만들지 않는 것이 중요. 높은 정밀도가 필요. 이때 정밀도를 지표로 확인

**민감도(sensitivity), 재현율(recall, TPRate), 진짜 양성 비율(TPR)**

- 전체 양성 데이터(TP + FN)중에 얼마나 많은 샘플을 양성으로 잘 분류했나?(TP)
- $\text{TP}/(\text{TP} + \text{FN})$

$$\text{민감도(recall, 재현율)} = \frac{\text{잘 예측(TP)}}{\text{전체 양성 샘플 전체(TP+FN)}}$$

- 재현율이 높아지면 FP는 상대적으로 낮아짐.
- 언제 사용? FN(가짜 음성. 잘못 예측함.)을 줄일 때, 성능 지표로 사용합니다.
- 재현율의 최적화와 정밀도의 최적화는 상충한다.

- 다른 말로 **민감도(sensitivity)**, **적중률(hit rate)**, **진짜 양성 비율(TPR)**이라고 합니다.
- 따라서 병원의 암 예측 같은 경우는 FN를 최소화시켜 재현율을 줄이면, 상대적으로 정밀도를 최대화된다.

## 특이도

- 실제 데이터의 음성 데이터(FP + TN)중에 제대로 예측한 샘플(음성 예측)?(TN)
- $TN / (FP + TN)$

$$\text{특이도} = \frac{\text{잘 예측}(TN)}{\text{실제 값이 음성인것 전체}(FP + TN)}$$

## FPRate

- 실제 데이터의 음성 데이터(FP + TN)중에 예측을 실패(양성 예측), 잘 분류하지 못한 것?(FP)
- $FP / (FP + TN)$

```
Wbegin{equation}
Wtext{FPRate} = Wfrac{Wtext{틀린 예측}(FP)}{Wtext{실제 값이 음성인것 전체}(FP + TN)}
Wend{equation}
```

## 다양한 분류 측정 방법

- [https://en.wikipedia.org/wiki/Sensitivity\\_and\\_specificity](https://en.wikipedia.org/wiki/Sensitivity_and_specificity)  
([https://en.wikipedia.org/wiki/Sensitivity\\_and\\_specificity](https://en.wikipedia.org/wiki/Sensitivity_and_specificity))
- 이진 분류에서는 정밀도와 재현율을 가장 많이 사용.
  - 분야마다 다른 지표를 사용할 수 있다.

## F-score

- 정밀도와 재현율은 중요한 측정 방법이지만, 둘중의 하나의 방법으로 전체 그림을 보기가 어렵다.
- 정밀도와 민감도(recall, 재현율)을 하나만 가지고 측정이 안된다. 정밀도(precision)와 재현율(recall)의 조화 평균인 f-점수 또는 f-측정은 이 둘을 하나로 요약해 줍니다.

$$F = 2 * \frac{\text{정밀도} * \text{재현율}}{\text{정밀도} + \text{재현율}}$$

위의 공식을 우리는  $f_1$  점수라고한다.

- 정밀도와 재현율을 함께 고려하므로 불균형한 이진 분류 데이터셋에서의 정확도보다 더 나은 지표가 될 수 있다.

## 3-3 f1-score를 확인해보기

## 각각의 모델 예측값을 f1-score로 예측

In [27]:

```
from sklearn.metrics import f1_score

# 빈도기반 모델 f1-score
print("무작위 더미 모델의 f1 score: {:.2f}".format(f1_score(y_test, pred_most_frequent)))

# Dummy분류 f1-score
print("무작위 더미 모델의 f1 score: {:.2f}".format(f1_score(y_test, pred_dummy)))

# 의사결정트리
print("트리 모델의 f1 score: {:.2f}".format(f1_score(y_test, pred_tree)))

# 로지스틱
print("로지스틱 회귀 모델의 f1 score: {:.2f}".format(f1_score(y_test, pred_logreg)))
```

무작위 더미 모델의 f1 score: 0.00  
 무작위 더미 모델의 f1 score: 0.00  
 트리 모델의 f1 score: 0.55  
 로지스틱 회귀 모델의 f1 score: 0.92

## f1-score를 요약해서 보여주기

- classification\_report(): 정밀도, 재현율, f1-score을 모두 한번에 계산
- support는 단순히 샘플의 수
- macro avg : 단순히 클래스별 점수의 평균 계산
- weighted avg : 클래스의 샘플수로 가중 평균

In [31]:

```
from sklearn.metrics import classification_report
print(classification_report(y_test, pred_most_frequent,
                           target_names=["not 9", "is 9"]))
```

	precision	recall	f1-score	support
not 9	0.90	1.00	0.94	403
is 9	0.00	0.00	0.00	47
accuracy			0.90	450
macro avg	0.45	0.50	0.47	450
weighted avg	0.80	0.90	0.85	450

## dummyClassifier 모델

In [32]:



```
from sklearn.metrics import classification_report
print(classification_report(y_test, pred_dummy,
                           target_names=["not 9", "is 9"]))
```

	precision	recall	f1-score	support
not 9	0.89	0.93	0.91	403
is 9	0.00	0.00	0.00	47
accuracy			0.83	450
macro avg	0.44	0.46	0.45	450
weighted avg	0.80	0.83	0.81	450

## 의사결정트리

In [34]:



```
from sklearn.metrics import classification_report
print(classification_report(y_test,
                           pred_tree,
                           target_names=["not 9", "is 9"]))
```

	precision	recall	f1-score	support
not 9	0.94	0.97	0.95	403
is 9	0.64	0.49	0.55	47
accuracy			0.92	450
macro avg	0.79	0.73	0.75	450
weighted avg	0.91	0.92	0.91	450

## 로지스틱 회귀

In [33]:



```
from sklearn.metrics import classification_report
print(classification_report(y_test,
                           pred_logreg,
                           target_names=["not 9", "is 9"]))
```

	precision	recall	f1-score	support
not 9	0.99	1.00	0.99	403
is 9	0.98	0.87	0.92	47
accuracy			0.98	450
macro avg	0.98	0.93	0.96	450
weighted avg	0.98	0.98	0.98	450

In [ ]:

