

# 핸즈온 머신러닝

## 3장. 분류

박해선(웁긴이)

[haesun.park@tensorflow.blog](mailto:haesun.park@tensorflow.blog)

<https://tensorflow.blog>

# MNIST dataset

- 머신러닝 분야의 ‘Hello world’ 문제입니다.
- 미국 고등학생과 인구조사국 직원들이 쓴 손글씨 숫자 70,000개
- <http://yann.lecun.com/exdb/mnist/>

Bunch 클래스

```
from sklearn.datasets import fetch_mldata
mnist = fetch_mldata('MNIST original')
mnist

{'DESCR': 'mldata.org dataset: mnist-original',
 'COL_NAMES': ['label', 'data'],
 'target': array([0., 0., 0., ..., 9., 9., 9.]),
 'data': array([[0, 0, 0, ..., 0, 0, 0],
                [0, 0, 0, ..., 0, 0, 0],
                [0, 0, 0, ..., 0, 0, 0],
                ...,
                [0, 0, 0, ..., 0, 0, 0],
                [0, 0, 0, ..., 0, 0, 0],
                [0, 0, 0, ..., 0, 0, 0]], dtype=uint8)}
```

# fetch\_mldata()

- mldata.org 사이트에서 매트랩 형식의 파일을 다운로드 합니다.
- ~/scikit\_learn\_data/mldata/ 에 저장합니다.
- mldata.org 사이트가 자주 다운됩니다. 예제를 진행하기 위해 mnist-original.mat 파일을 수동으로 다운로드하여 ~/scikit\_learn\_data/mldata/ 에 옮겨 놓습니다.
- <https://github.com/amplab/datascience-sp14/raw/master/lab7/mldata/mnist-original.mat>

# mnist preview

```
X, y = mnist["data"], mnist["target"]  
X.shape
```

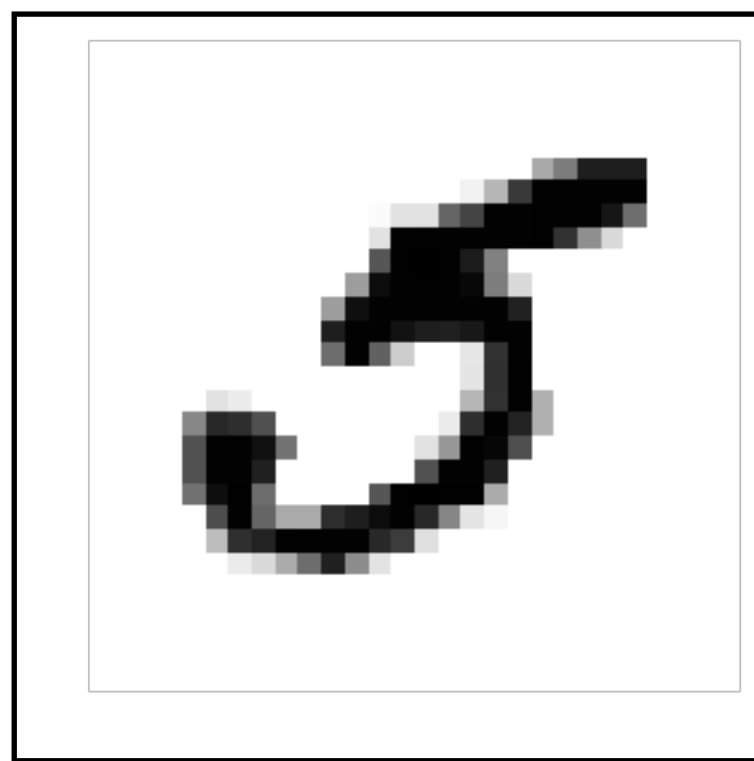
```
(70000, 784)
```

```
y.shape
```

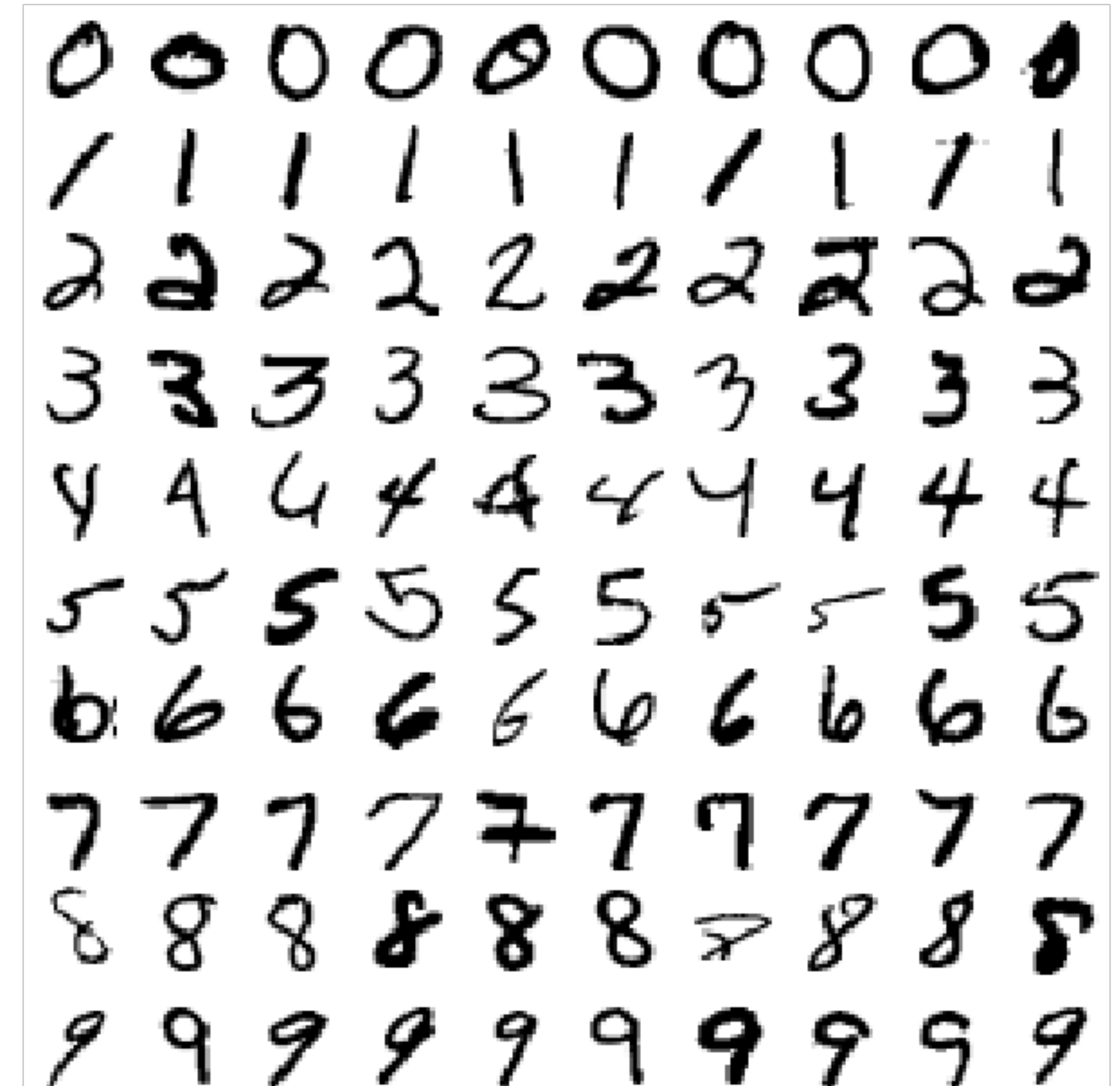
```
(70000,)
```

```
some_digit = X[36000]
```

28



28



# Train & Test set

- MNIST 데이터셋은 60,000개를 훈련 세트로 10,000개를 테스트 세트로 사용합니다
- mnist는 숫자 순서대로 데이터가 나열되어 있습니다. 알고리즘에 따라 문제가 될 수 있으므로 데이터를 무작위로 섞습니다(ex, SGD).

```
X_train, X_test, y_train, y_test = X[:60000], X[60000:], y[:60000], y[60000:]
```

```
import numpy as np
```

```
shuffle_index = np.random.permutation(60000)
```

```
X_train, y_train = X_train[shuffle_index], y_train[shuffle_index]
```

0~59,999 사이의 랜덤한 수열을 만듭니다

SGDClassifier 같은 경우 기본적으로 데이터를 섞습니다.  
하지만 아직 어떤 알고리즘을 사용할지 모릅니다.



# 이진(5 vs not-5) 분류

```
y_train_5 = (y_train == 5)
y_test_5 = (y_test == 5)
```

0~9까지 타겟이 0(not-5)~1(5)로 바꿉니다  
0: 54,579개 1: 5,421개

```
from sklearn.linear_model import SGDClassifier
```

확률적 경사 하강법 분류 모델

```
sgd_clf = SGDClassifier(max_iter=5, random_state=42)
sgd_clf.fit(X_train, y_train_5)
```

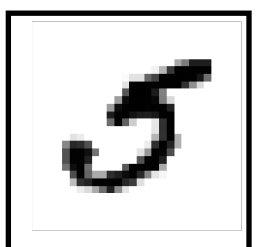
max\_iter(epoch) 기본값

```
SGDClassifier(alpha=0.0001, average=False, class_weight=None, epsilon=0.1,
              eta0=0.0, fit_intercept=True, l1_ratio=0.15,
              learning_rate='optimal', loss='hinge', max_iter=5, n_iter=None,
              n_jobs=1, penalty='l2', power_t=0.5, random_state=42, shuffle=True,
              tol=None, verbose=0, warm_start=False)
```

```
sgd_clf.predict([some_digit])
```

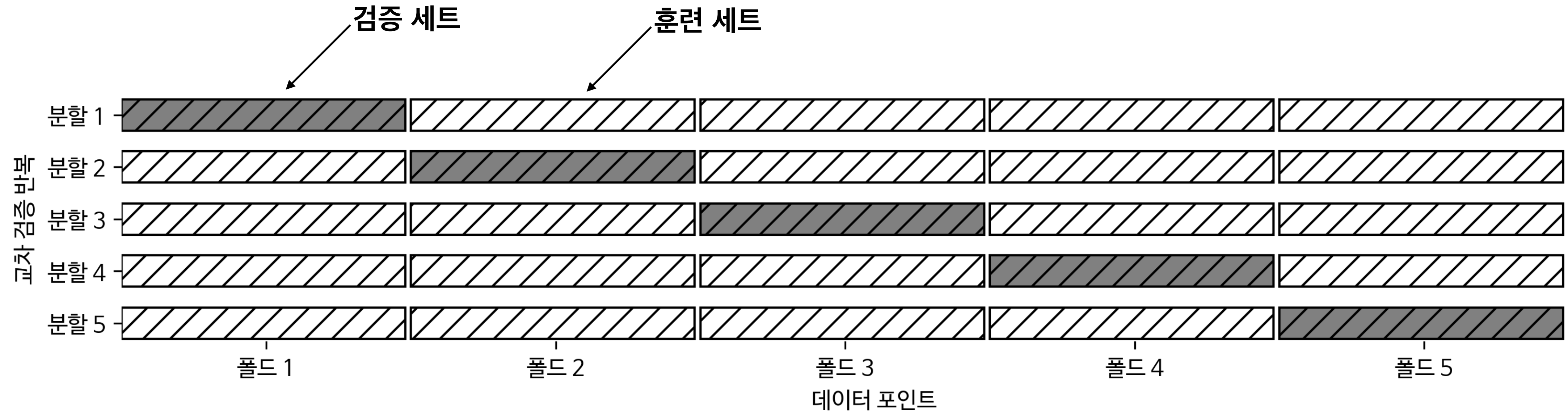
some\_digit = X[36000]

```
array([ True])
```



# 교차 검증

- 훈련 세트 중 일부를 검증(개발) 세트로 사용하여 모델의 성능을 추정합니다.
- 일반화 성능을 왜곡하지 않으려고 테스트 세트를 사용하지 않습니다(테스트 세트에 과대적합을 피하려고 혹은 테스트 세트의 정보 누설을 막으려고)



# sgd\_clf 교차검증

```
from sklearn.model_selection import cross_val_score
cross_val_score(sgd_clf, X_train, y_train_5, cv=3, scoring="accuracy")
```

`cross_validate()`  
3개의 폴드, 정확도 기준

```
array([0.9502 , 0.96565, 0.96495])
```

SGD 분류기는 대략 95% 성능이 기대됩니다

```
from sklearn.base import BaseEstimator
class Never5Classifier(BaseEstimator):
    def fit(self, X, y=None):
        pass
    def predict(self, X):
        return np.zeros((len(X), 1), dtype=bool)
```

무조건 0으로 예측하는 분류기

(len(X), 1) 크기의 0으로 채워진 배열

```
never_5_clf = Never5Classifier()
cross_val_score(never_5_clf, X_train, y_train_5, cv=3, scoring="accuracy")
```

```
array([0.909 , 0.90715, 0.9128 ])
```

KFold 사용

기본값

X\_train의 90% 정도가 0 샘플이므로(불균형)  
무조건 0으로만 예측해도 90% 성능 달성

```
from sklearn.dummy import DummyClassifier
never_5_dummy = DummyClassifier(strategy='most_frequent')
never_5_dummy.fit(X_train, y_train_5)
cross_val_score(never_5_dummy, X_train, y_train_5)
```

most\_frequent: 가장 많은 클래스 레이블로 예측

```
array([0.90965, 0.90965, 0.90965])
```

StratifiedKFold 사용



# 오차 행렬

- 불균형 데이터셋은 정확도만으로 평가하기 어렵습니다
- 오차행렬(confusion matrix)은 분류 모델의 성능을 평가하기 위해 사용합니다

```
from sklearn.metrics import confusion_matrix
```

```
y_train_pred_no_cv = sgd_clf.predict(X_train)  
confusion_matrix(y_train_5, y_train_pred_no_cv)
```

```
array([[53470, 1109],  
       [ 1003, 4418]])
```

음성 클래스	TN	FP
양성 클래스	FN	TP
	음성 예측	양성 예측

# 오차 행렬 비교

```
from sklearn.model_selection import cross_val_predict
```

```
y_train_pred = cross_val_predict(sgd_clf, X_train, y_train_5, cv=3)
```

```
confusion_matrix(y_train_5, y_train_pred)
```

```
array([[53272, 1307],  
       [ 1077, 4344]])
```

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

음성 클래스	TN	FP
양성 클래스	FN	TP
	음성 예측	양성 예측

```
y_train_pred_dummy = cross_val_predict(never_5_dummy, X_train, y_train_5)  
confusion_matrix(y_train_5, y_train_pred_dummy)
```

```
array([[54579, 0],  
       [ 5421, 0]])
```

확실히 뭔가 문제가 있군요  
불균형 데이터셋에서 정확도는 좋은 지표가 아닙니다

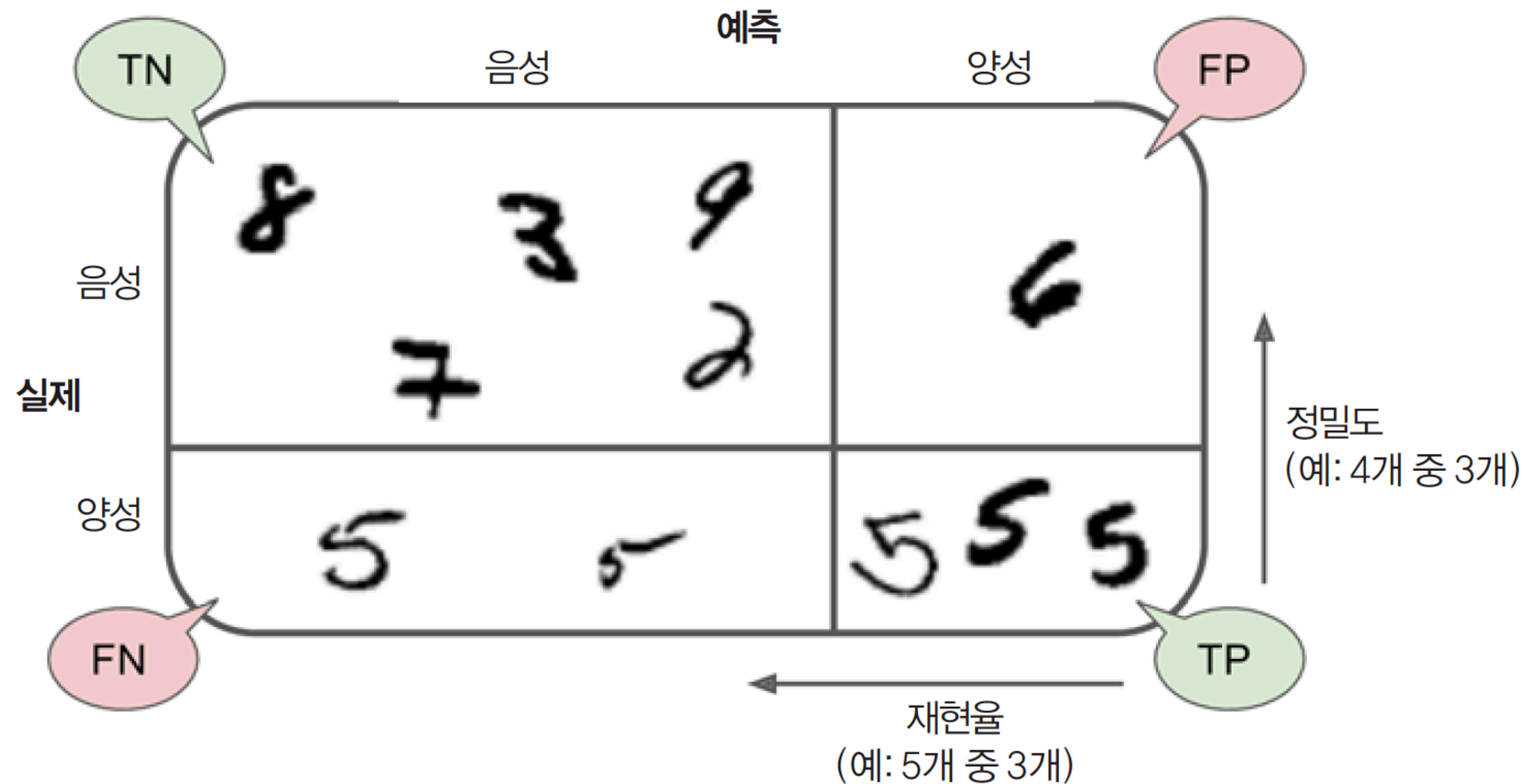
# 정밀도와 재현율

$$\text{정밀도} = \frac{TP}{TP + FP}$$

$$\text{재현율} = \frac{TP}{TP + FN}$$

정밀도(precision)=양성예측도(PPV)

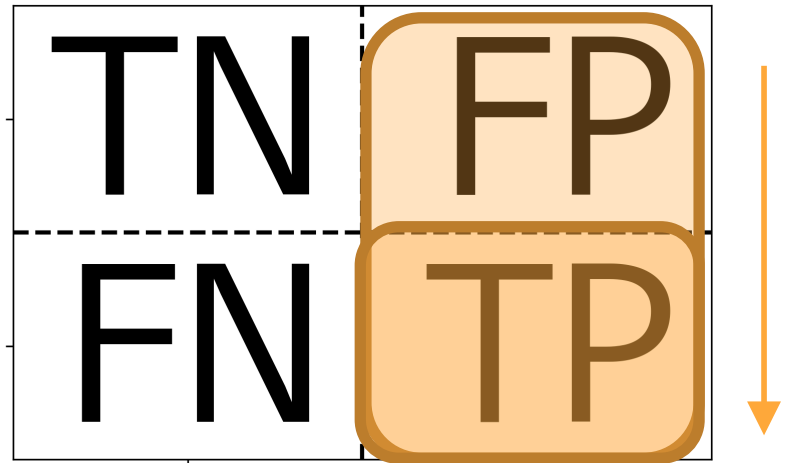
재현율(recall)=민감도(sensitivity)=진짜양성비율(TPR)



# 정밀도 비교

$$\text{정밀도} = \frac{TP}{TP + FP}$$

TN	FP
FN	TP



```
from sklearn.metrics import precision_score, recall_score  
precision_score(y_train_5, y_train_pred)
```

0.7687135020350381

```
4344 / (4344 + 1307)
```

0.7687135020350381

```
array([[53272, 1307],  
       [ 1077, 4344]])
```

```
precision_score(y_train_5, y_train_pred_dummy)
```

```
array([[54579, 0],  
       [ 5421, 0]])
```

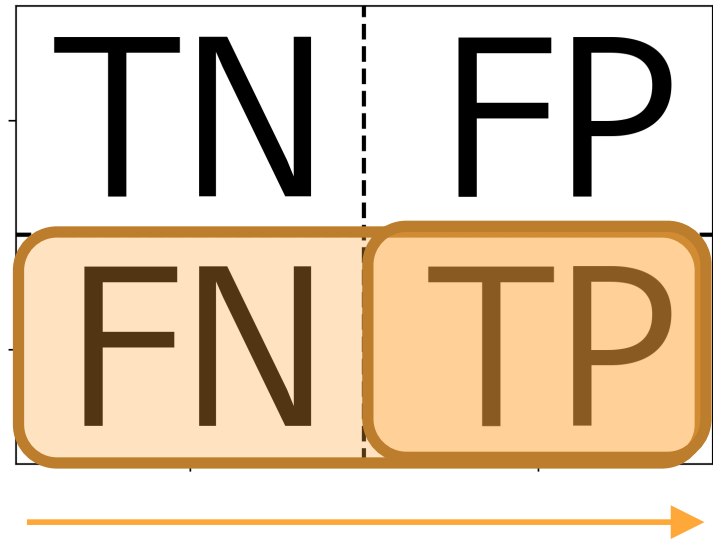
```
/home/haesun/anaconda3/envs/hands-on-ml/lib/python3.6/site-packa  
tion.py:1135: UndefinedMetricWarning: Precision is ill-defined  
predicted samples.
```

```
'precision', 'predicted', average, warn_for)
```

0.0

# 재현율 비교

$$\text{재현율} = \frac{TP}{TP + FN}$$



```
recall_score(y_train_5, y_train_pred)
```

```
0.801328168234643
```

```
4344 / (4344 + 1077)
```

```
0.801328168234643
```

```
array([[53272, 1307],  
       [ 1077, 4344]])
```

```
recall_score(y_train_5, y_train_pred_dummy)
```

```
0.0
```

```
array([[54579, 0],  
       [ 5421, 0]])
```

정밀도와 재현율을 보면 확실히 잘못된 것을 알 수 있습니다

[https://en.wikipedia.org/wiki/Precision\\_and\\_recall](https://en.wikipedia.org/wiki/Precision_and_recall)



# f1-score

- f1-점수는 정밀도와 재현율의 조화 평균입니다.

$$F = \frac{1}{\frac{\alpha}{\text{정밀도}} + \frac{1-\alpha}{\text{재현율}}} = (\beta^2 + 1) \frac{\text{정밀도} \times \text{재현율}}{\beta^2 \times \text{정밀도} + \text{재현율}} \quad \beta^2 = \frac{1-\alpha}{\alpha}$$

$$F_1 = \frac{2}{\frac{1}{\text{정밀도}} + \frac{1}{\text{재현율}}} = 2 \times \frac{\text{정밀도} \times \text{재현율}}{\text{정밀도} + \text{재현율}}$$

# f1-score 비교

```
from sklearn.metrics import f1_score  
f1_score(y_train_5, y_train_pred)
```

0.7846820809248555

정확도는 95% 정도지만 f1-점수는 낮습니다

```
f1_score(y_train_5, y_train_pred_dummy)
```

```
/home/haesun/anaconda3/envs/hands-on-ml/lib/python3.6/site-packages/sklearn/metrics/_classification.py:1135: UndefinedMetricWarning: F-score is ill-defined for  
predicted samples.  
  'precision', 'predicted', average, warn_for)
```

0.0

# classification\_report

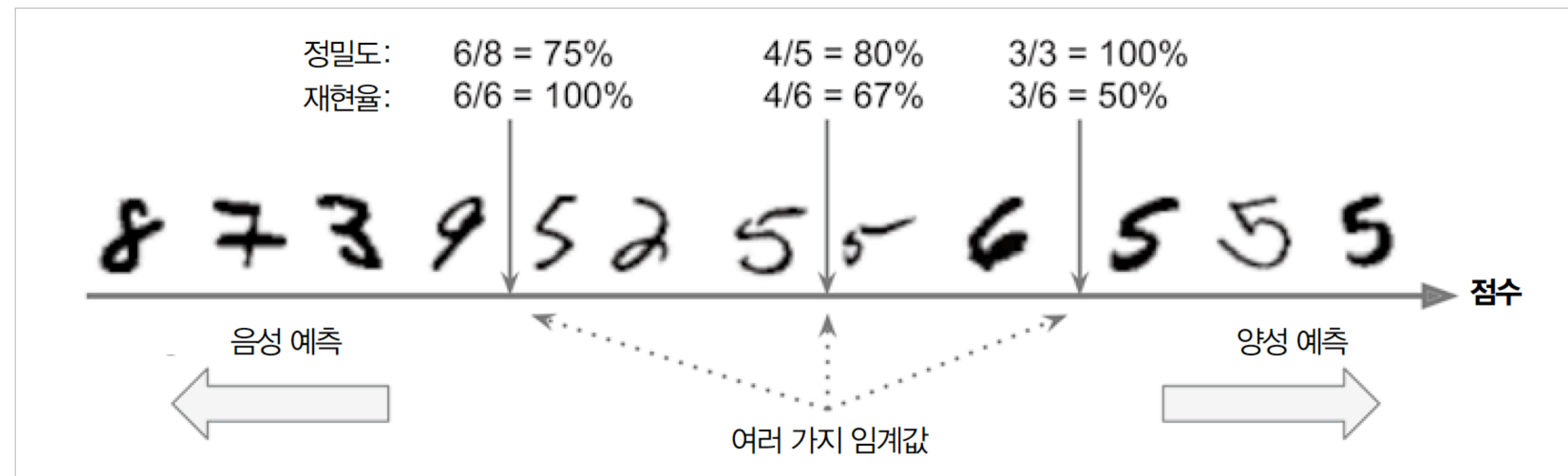
- 정밀도, 재현율, f1-점수를 한번에 출력합니다

```
from sklearn.metrics import classification_report
print(classification_report(y_train_5, y_train_pred))
```

		precision	recall	f1-score	support	
해당 클래스가 양성일 때 통갯값	False	0.98	0.98	0.98	54579	
	True	0.77	0.80	0.78	5421	
avg / total		0.96	0.96	0.96	60000	샘플 개수를 고려하여 가중 평균합니다

# 정밀도/재현율 트레이드오프

- 정밀도를 올리면 재현율이 줄고 그 반대도 마찬가지입니다.
- 안전한 동영상을 판단하는 분류기는 정밀도를 높아야 합니다(안전한 영상만 양성으로 판단해야 합니다). 재현율은 조금 낮아도 됩니다(안전한 영상이 유해 영상으로 분류되더라도).
- 도둑을 잡는 감시 카메라는 정밀도가 낮더라도(도둑이 아닌데 경보를 울리더라도) 재현율이 높아야 합니다(모든 도둑은 반드시 잡아야 합니다).



# decision\_function()

- 양성 예측에 대한 확신이 높을수록 decision\_function()의 값이 높습니다

```
y_scores = sgd_clf.decision_function([some_digit])  
y_scores  
  
array([161855.74572176])
```

```
threshold = 0  
y_some_digit_pred = (y_scores > threshold)
```

결정함수 값이 0보다 클 때 양성으로 판단합니다

```
y_some_digit_pred  
  
array([ True])
```

```
threshold = 200000  
y_some_digit_pred = (y_scores > threshold)  
y_some_digit_pred
```

결정함수 값이 200,000보다 클 때 양성으로 판단합니다

```
array([False])
```

음성으로 예측합니다.



# precision\_recall\_curve()

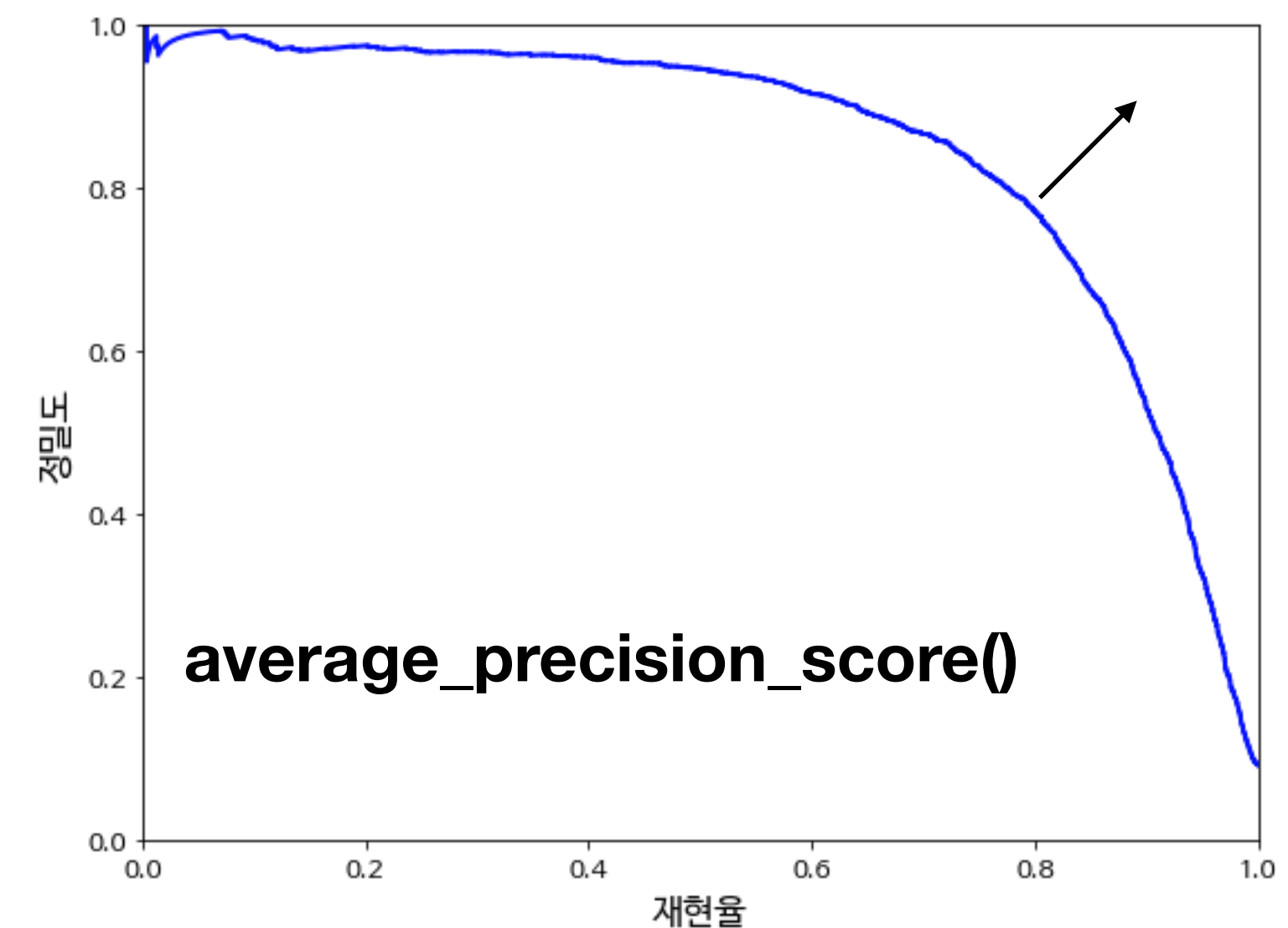
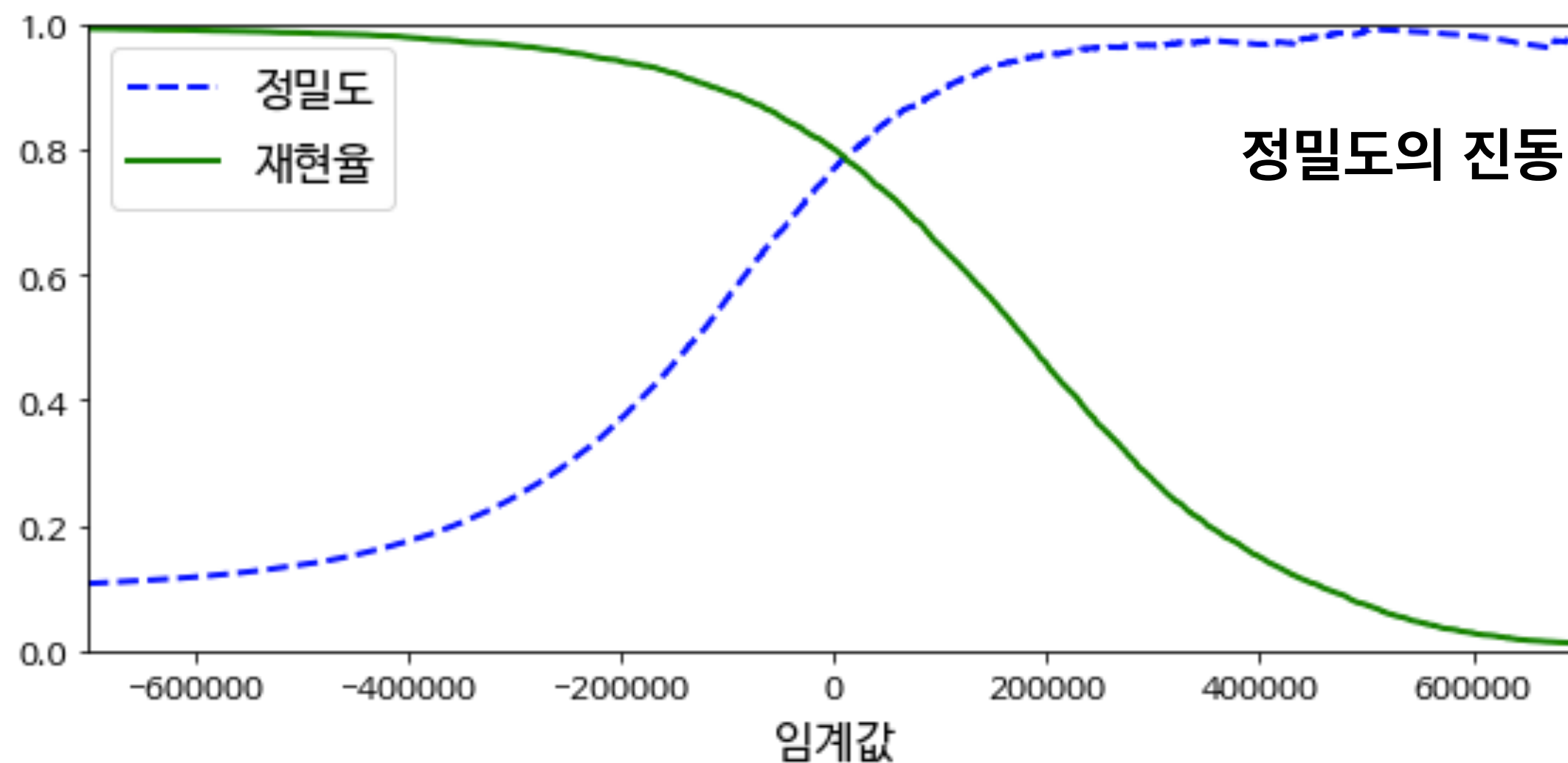
- 임계값에 따라 정밀도와 재현율 값을 반환합니다

(50000, ) 차원  
배열 스칼라(array scalar)

```
y_scores = cross_val_predict(sgd_clf, X_train, y_train_5, cv=3,  
                             method="decision_function")
```

method='predict' 기본값,  
'predict\_proba'도 가능

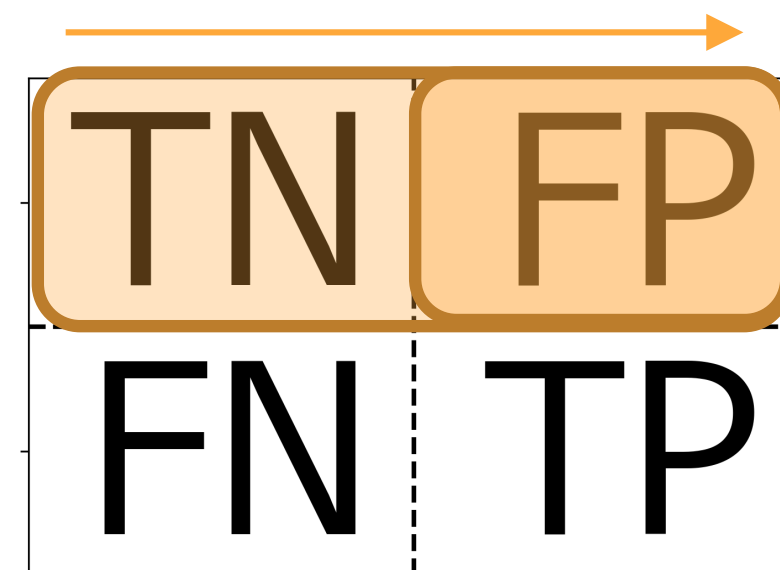
```
from sklearn.metrics import precision_recall_curve  
  
precisions, recalls, thresholds = precision_recall_curve(y_train_5, y_scores)
```



# 거짓양성비율(FPR)

- 전체 음성 샘플 중 거짓 양성으로 분류된 비율

$$\text{FPR} = \frac{FP}{TN + FP}$$



$$= 1 - \frac{TN}{FP + TN} = 1 - \text{TNR(특이도)}$$

전체 음성 샘플 중 음성으로 분류된 비율(True Negative Rate)

# ROC curve

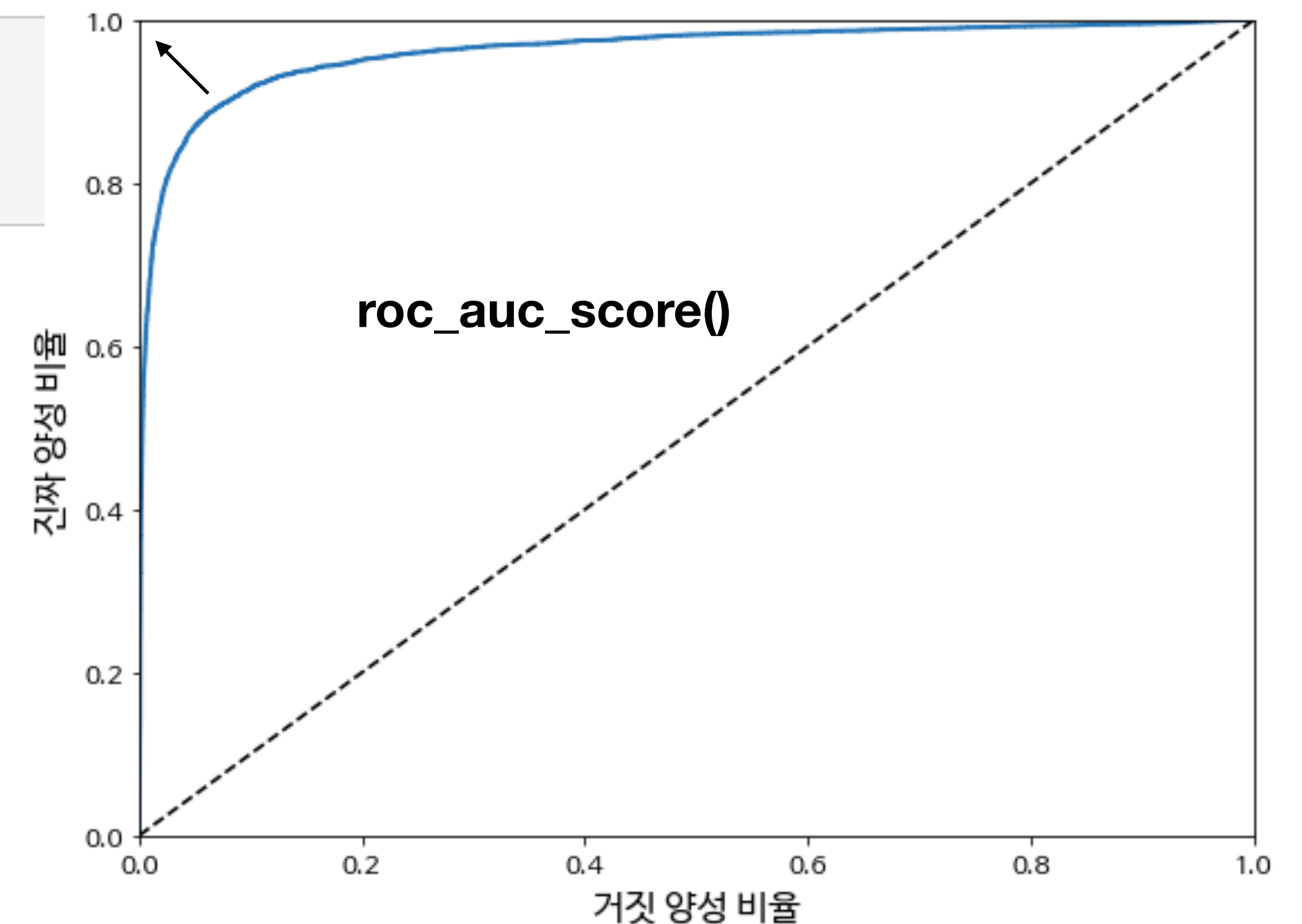
- ROC(수신기 조작 특성) 곡선은 FPR(거짓양성비율)에 대한 TPR(진짜양성비율, 재현율) 곡선입니다

```
from sklearn.metrics import roc_curve  
  
fpr, tpr, thresholds = roc_curve(y_train_5, y_scores)
```

결정함수 값

```
from sklearn.metrics import roc_auc_score  
  
roc_auc_score(y_train_5, y_scores)
```

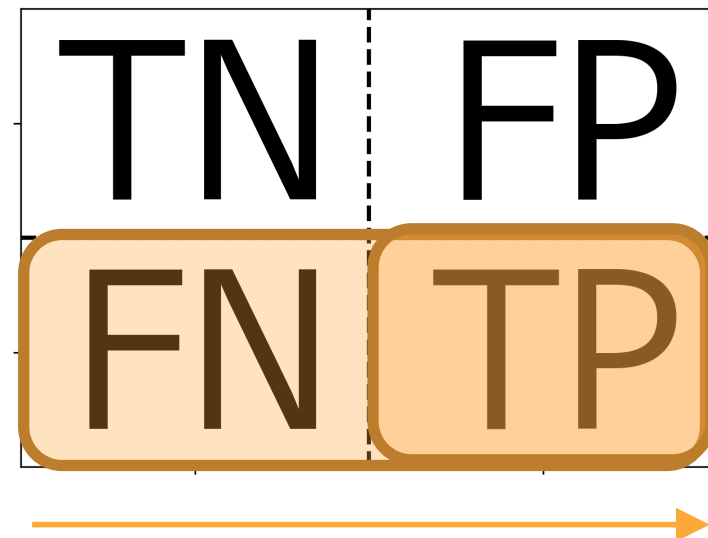
0.9624496555967155



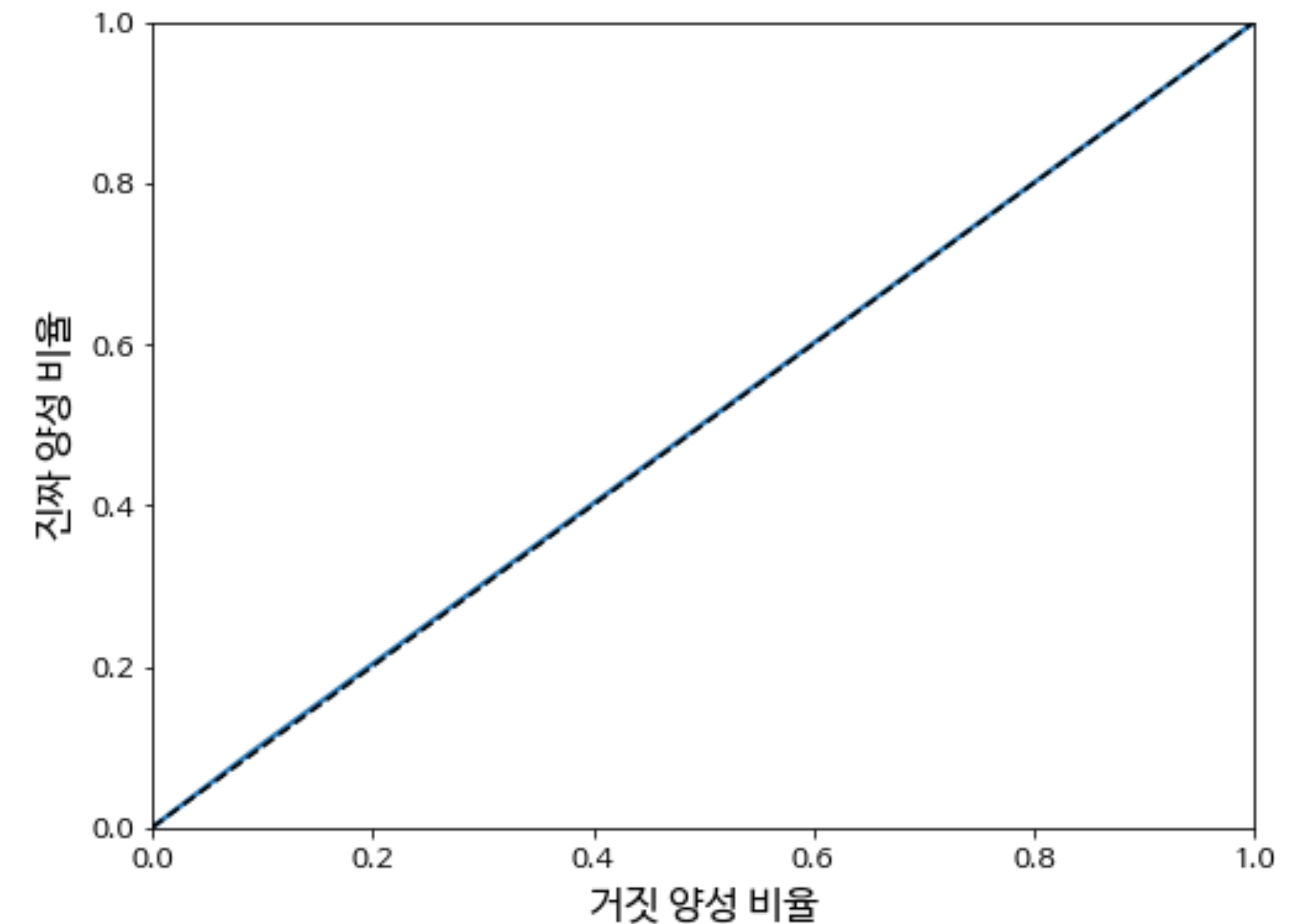
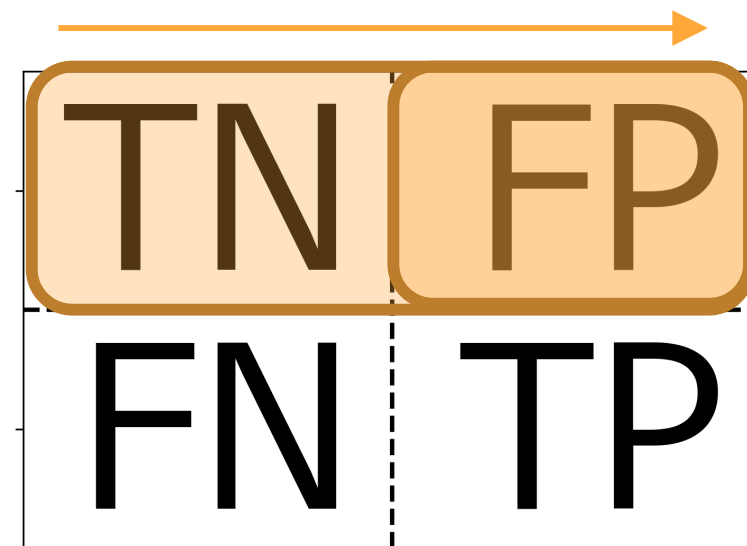
# dummy ROC curve

```
random_dummy = DummyClassifier(strategy='stratified')
random_dummy.fit(X_train, y_train_5)
random_y_scores = cross_val_predict(random_dummy, X_train, y_train_5, method='predict_proba')
fpr, tpr, thresholds = roc_curve(y_train_5, random_y_scores[:, 1])
```

$$\text{재현율} = \frac{TP}{TP + FN}$$



$$\text{FPR} = \frac{FP}{TN + FP}$$



# 다중 분류

- 로지스틱 회귀, 트리 기반 모델을 제외하면 대부분 이진 분류만을 지원합니다
- 클래스 하나와 나머지를 분류하는 방법인 일대다 One-versus-All, One-versus-Rest 방식 (LinearSVC, SVC, SGDClassifier, OneVsRestClassifier, ...)

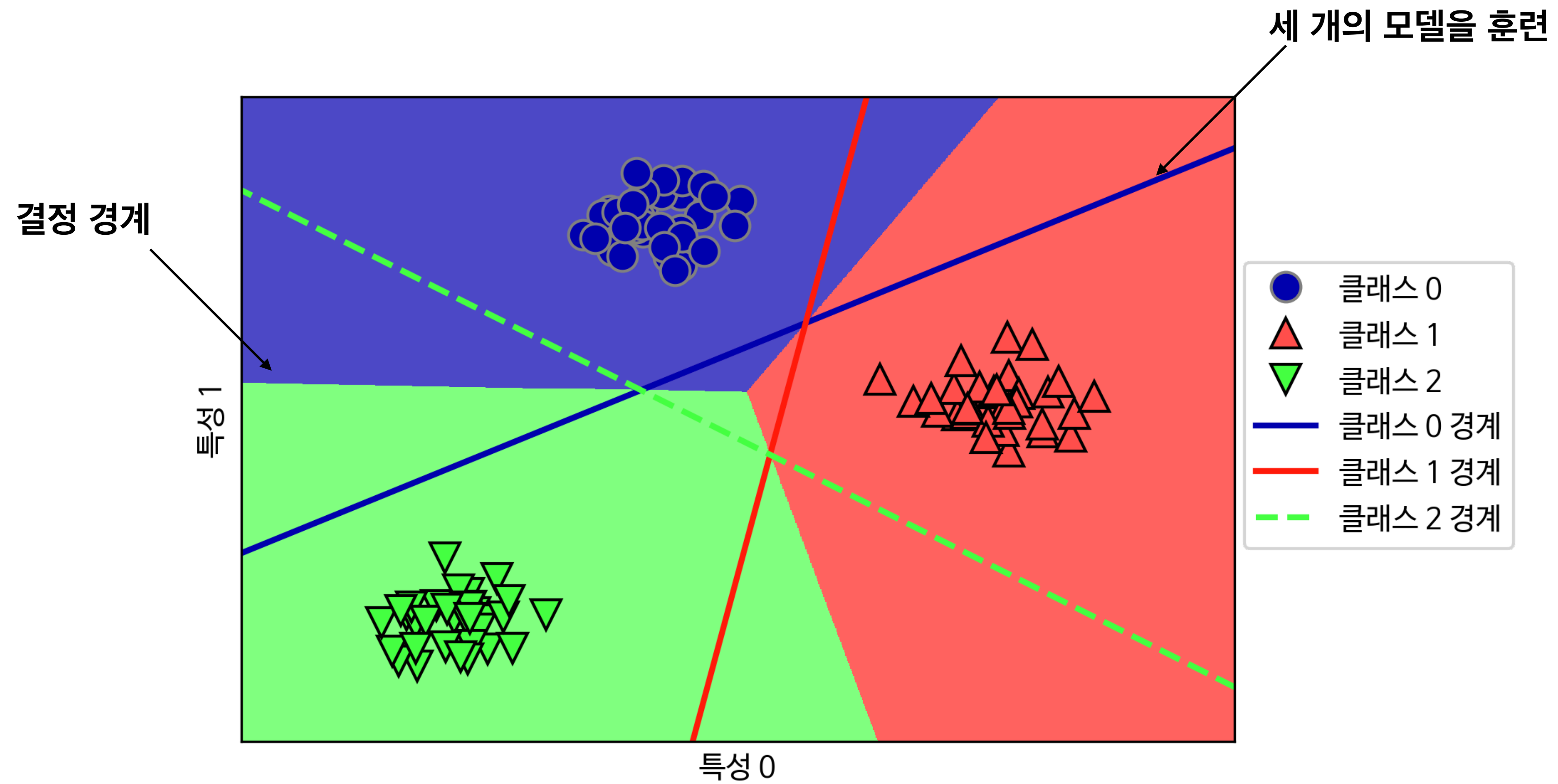
필요한 분류기 개수:  $n$

- 두 개의 클래스 마다 분류기를 훈련하는 일대일 One-versus-One 방식 (SVC, OneVsOneClassifier)

필요한 분류기 개수:  $\binom{n}{2} = \frac{n!}{2!(n-2)!} = \frac{n \times (n-1)}{2}$



# One-versus-All



# MNIST 다중 분류

```
sgd_clf.fit(X_train, y_train)
sgd_clf.predict([some_digit])
```

원본 타깃 데이터 사용

```
array([5.])
```

```
some_digit_scores = sgd_clf.decision_function([some_digit])
some_digit_scores
```

```
array([[ -311402.62954431, -363517.28355739, -446449.5306454 ,
        -183226.61023518, -414337.15339485,  161855.74572176,
        -452576.39616343, -471957.14962573, -518542.33997148,
        -536774.63961222]])
```

클래스마다 결정함수 값이 계산됩니다

```
np.argmax(some_digit_scores)
```

가장 큰 인덱스가 예측에 사용됩니다

```
5
```

```
sgd_clf.classes_
```

```
array([0., 1., 2., 3., 4., 5., 6., 7., 8., 9.])
```

```
sgd_clf.classes_[5]
```

클래스의 인덱스는 클래스 값이 아닙니다

```
5.0
```

# OneVsOneClassifier

```
from sklearn.multiclass import OneVsOneClassifier
ovo_clf = OneVsOneClassifier(SGDClassifier(max_iter=5, random_state=42))
ovo_clf.fit(X_train, y_train)
ovo_clf.predict([some_digit])
```

array([5.])

```
len(ovo_clf.estimators_)
```

45

$$\binom{10}{2} = \frac{10!}{2!(10-2)!} = 45$$

```
forest_clf.fit(X_train, y_train)
forest_clf.predict([some_digit])
```

array([5.])

predict() 메서드는 클래스 값을 반환합니다

```
forest_clf.predict_proba([some_digit])
```

array([[0.1, 0. , 0. , 0.1, 0. , 0.8, 0. , 0. , 0. , 0. ]])

예측 신뢰도를 확인할 수 있습니다

# 다중 분류 교차 검증

```
cross_val_score(sgd_clf, X_train, y_train, cv=3, scoring="accuracy")
```

```
array([0.84063187, 0.84899245, 0.86652998])
```

무조건 하나의 클래스를 예측했다면 10% 정확도를 얻습니다

```
from sklearn.preprocessing import StandardScaler
```

```
scaler = StandardScaler()
```

```
X_train_scaled = scaler.fit_transform(X_train.astype(np.float64))
```

```
cross_val_score(sgd_clf, X_train_scaled, y_train, cv=3, scoring="accuracy")
```

평균을 0, 분산을 1로 바꿉니다

```
array([0.91011798, 0.90874544, 0.906636  ])
```



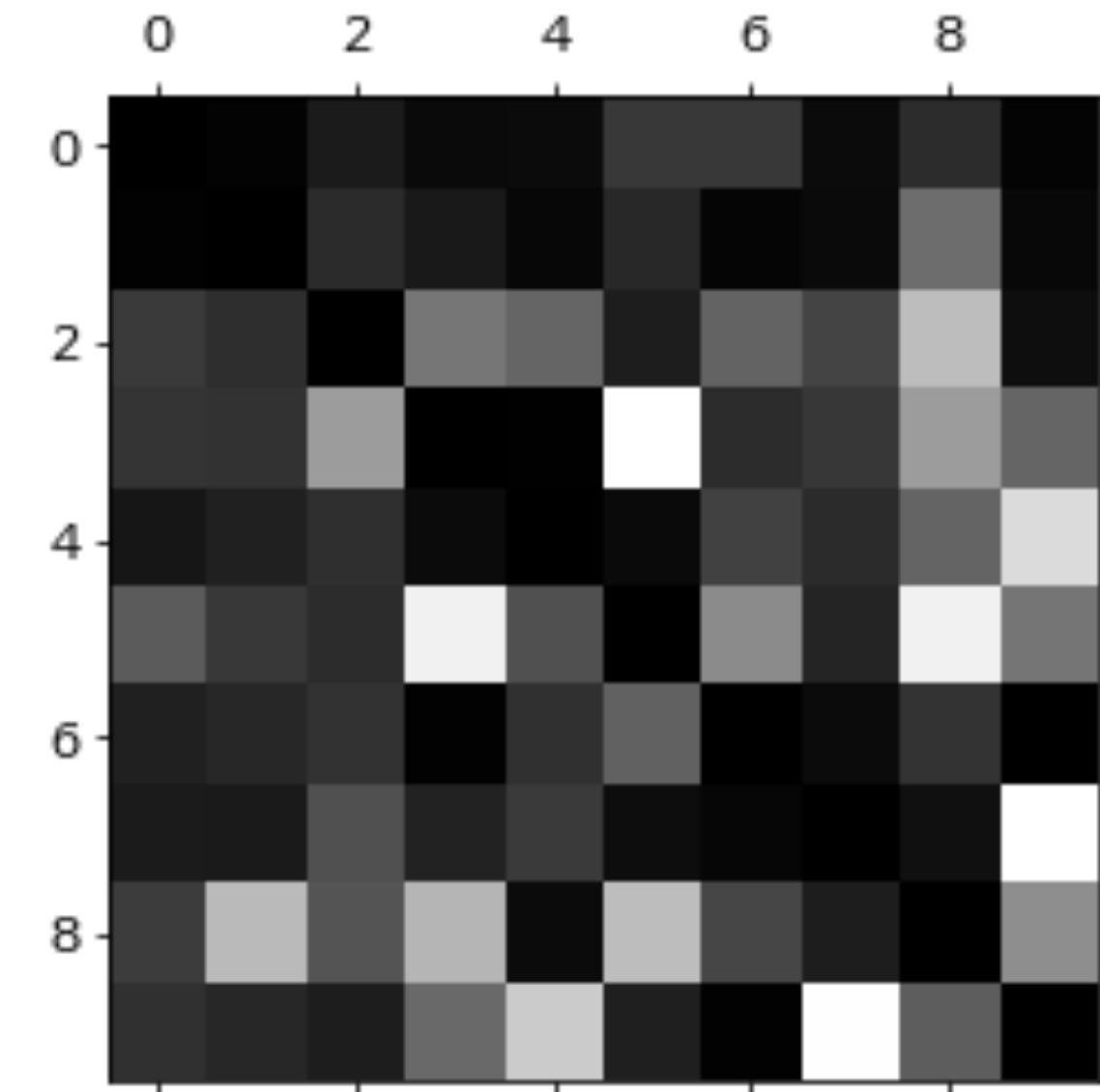
# 다중 분류 오차 행렬

```
y_train_pred = cross_val_predict(sgd_clf, X_train_scaled, y_train, cv=3)
conf_mx = confusion_matrix(y_train, y_train_pred)
conf_mx
```

```
array([[5725,   3,  24,   9,  10,  49,  50,  10,  39,   4],
       [   2, 6493,  43,  25,   7,  40,   5,  10, 109,   8],
       [  51,  41, 5321, 104,  89,  26,  87,  60, 166,  13],
       [  47,  46, 141, 5342,   1, 231,  40,  50, 141,  92],
       [  19,  29,  41,  10, 5366,   9,  56,  37,  86, 189],
       [  73,  45,  36, 193,  64, 4582, 111,  30, 193,  94],
       [  29,  34,  44,   2,  42,  85, 5627,  10,  45,   0],
       [  25,  24,  74,  32,  54,  12,   6, 5787,  15, 236],
       [  52, 161,  73, 156,  10, 163,  61,  25, 5027, 123],
       [  43,  35,  26,  92, 178,  28,   2, 223,  82, 5240]])
```

```
row_sums = conf_mx.sum(axis=1, keepdims=True)
norm_conf_mx = conf_mx / row_sums
```

행별 비율로 히트맵을 만듭니다





# 다중 분류 리포트

```
print(classification_report(y_train, y_train_pred))
```

	precision	recall	f1-score	support
0.0	0.94	0.97	0.96	5923
1.0	0.94	0.96	0.95	6742
2.0	0.91	0.89	0.90	5958
3.0	0.90	0.87	0.88	6131
4.0	0.92	0.92	0.92	5842
5.0	0.88	0.85	0.86	5421
6.0	0.93	0.95	0.94	5918
7.0	0.93	0.92	0.93	6265
8.0	0.85	0.86	0.86	5851
9.0	0.87	0.88	0.88	5949
avg / total	0.91	0.91	0.91	60000

# RandomForestClassifier

```
from sklearn.ensemble import RandomForestClassifier
forest_clf = RandomForestClassifier(random_state=42)
y_probas_forest = cross_val_predict(forest_clf, X_train, y_train_5, cv=3,
                                     method="predict_proba")
```

```
y_scores_forest = y_probas_forest[:, 1] # 점수는 양성 클래스의 확률입니다
fpr_forest, tpr_forest, thresholds_forest = roc_curve(y_train_5, y_scores_forest)
```

```
roc_auc_score(y_train_5, y_scores_forest)
```

0.9931243366003829

0.962

```
y_train_pred_forest = cross_val_predict(forest_clf, X_train, y_train_5, cv=3)
precision_score(y_train_5, y_train_pred_forest)
```

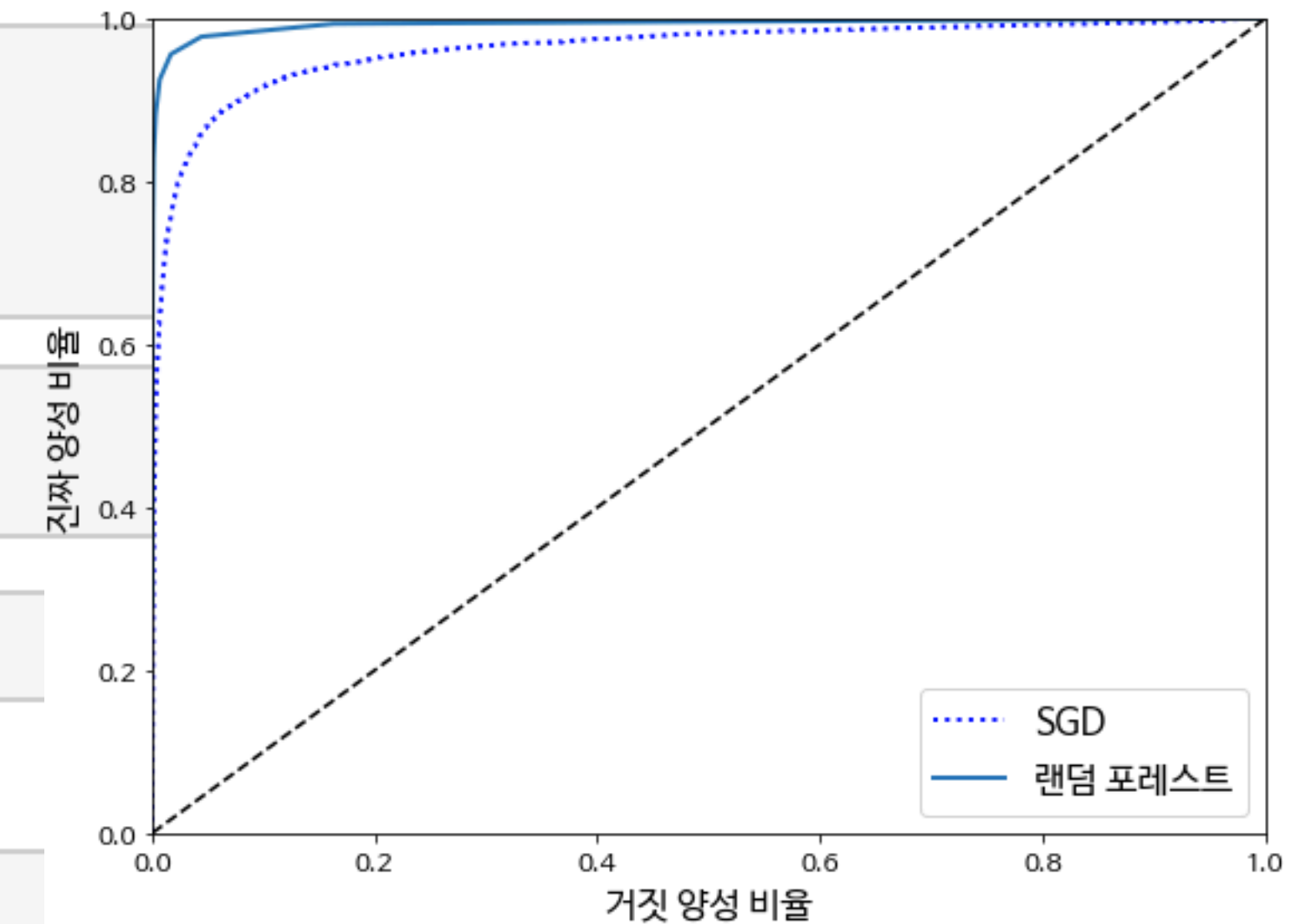
0.9852973447443494

0.768

```
recall_score(y_train_5, y_train_pred_forest)
```

0.8282604685482383

0.801



# 다중 레이블 분류

- 여러개의 이진 레이블을 출력합니다(ex, 사진에 밥, 엘리스, 찰리 등장 여부)

```
from sklearn.neighbors import KNeighborsClassifier
```

```
y_train_large = (y_train >= 7)
y_train_odd = (y_train % 2 == 1)
y_multilabel = np.c_[y_train_large, y_train_odd]
```

```
knn_clf = KNeighborsClassifier()
knn_clf.fit(X_train, y_multilabel)
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=1, n_neighbors=5, p=2,
                     weights='uniform')
```

```
knn_clf.predict([some_digit])
```

```
array([[False,  True]])
```

경고: 다음 셀은 실행하는데 매우 오래 걸립니다(하드웨어에 따라 몇 시간이 걸릴 수 있습니다).

```
y_train_knn_pred = cross_val_predict(knn_clf, X_train, y_multilabel, cv=3, n_jobs=-1)
f1_score(y_multilabel, y_train_knn_pred, average="macro")
```

```
0.97709078477525
```

DecisionTreeClassifier, RandomForestClassifier,  
MLPClassifier

두 개의 레이블: 7보다 큰지와 홀수 여부

두 개의 레이블을 예측합니다

macro: 클래스별 통계 평균  
micro: 전체 레이블 합산 평균  
weighted: 샘플 수 가중치 평균

# classification\_report

```
print(classification_report(y_multilabel, y_train_knn_pred))
```

	precision	recall	f1-score	support
0	0.98	0.97	0.97	18065
1	0.98	0.99	0.98	30508
avg / total	0.98	0.98	0.98	48573

**average='weighted' 적용**



# 다중 출력 분류

- 다중 출력 다중 클래스 분류라고도 부릅니다. 다중 레이블 분류에서 이진 클래스를 다중 클래스로 확장한 것입니다

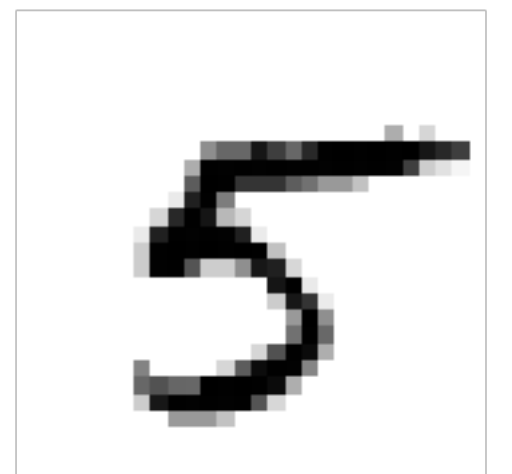
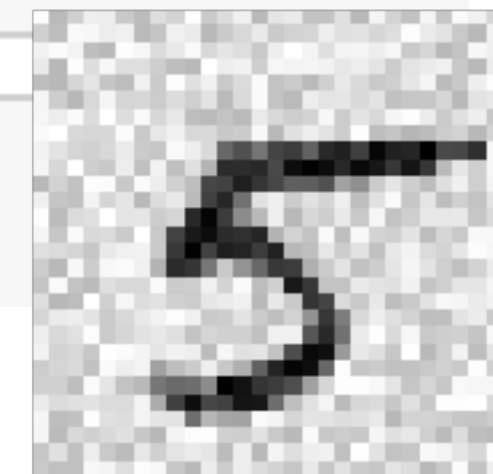
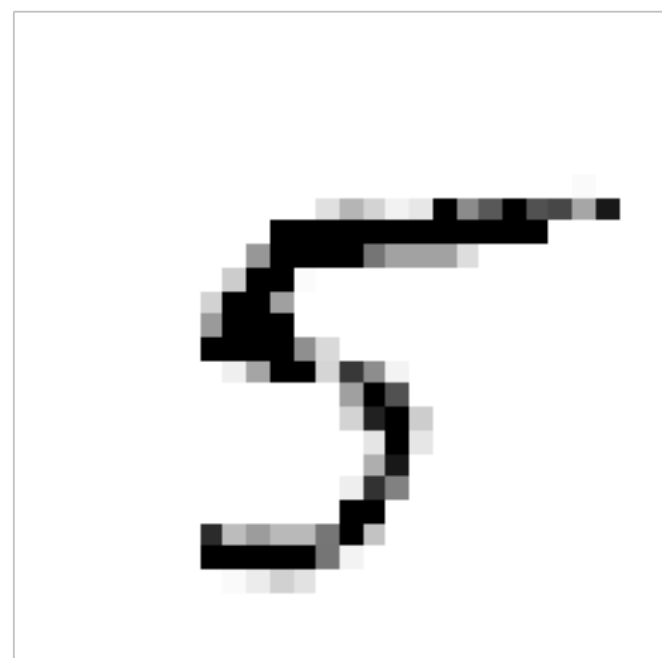
```
noise = np.random.randint(0, 100, (len(X_train), 784))
X_train_mod = X_train + noise
noise = np.random.randint(0, 100, (len(X_test), 784))
X_test_mod = X_test + noise
y_train_mod = X_train
y_test_mod = X_test
```

훈련 데이터: 노이즈 삽입된 이미지

테스트 데이터: 원본 이미지

다중 레이블: 784개 픽셀  
다중 출력: 0~255 픽셀 값

```
knn_clf.fit(X_train_mod, y_train_mod)
clean_digit = knn_clf.predict([X_test_mod[some_index]])
```



DecisionTreeClassifier, RandomForestClassifier

감사합니다