3장

분류

MNIST dataset

- 머신러닝 분야의 'Hello world' 문제
- 미국 고등학생과 인구조사국 직원들이 쓴 손글씨 숫자 70,000개

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

mldata.org 사이트에서 매트랩 형식의 파일을 다운로드해서 ~/scikit_learn_data/mldata/ 에 저장함

```
{'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)}
```

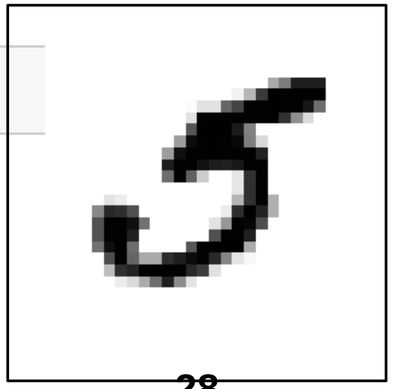
mnist preview

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

```
(70000, 784)
```

```
y.shape
```

```
(70000,)
```




```
%matplotlib inline
import matplotlib
import matplotlib.pyplot as plt
some_digit = X[36000]
some_digit_image = some_digit.reshape(28, 28)
plt.imshow(some_digit_image, cmap = matplotlib.cm.binary,
   interpolation="nearest")
plt.axis("off")
plt.show()
```

```
>>> y[36000]
5.0
```

Train, Test set 준비

- MNIST 데이터셋은 60,000개를 훈련 세트로 10,000개를 테스트 세트로 나누어 놓았음
- mnist는 숫자 순서대로 데이터가 나열되어 있습니다. 학습데이터를 무작위로 섞어 놓기
  (SGDclassifier경우는 시작 전에 섞어 놓지만 다른 알고리즘은 안 그러니 속 편하게 미리 섞어 놓자)

```
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]
```

이진(5vs not-5) 분류 : 데이터 준비 + 훈련

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

0~9까지 타깃을 False(not-5) 혹은 True(5)로 변경
False: 54,579개 True: 5,421개

```
from sklearn.linear_model import SGDClassifier

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

확률적 경사 하강법 분류 모델로 분류

```
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)
```

some_digit = X[36000]

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

some_digit를 학습한 모델로 추론

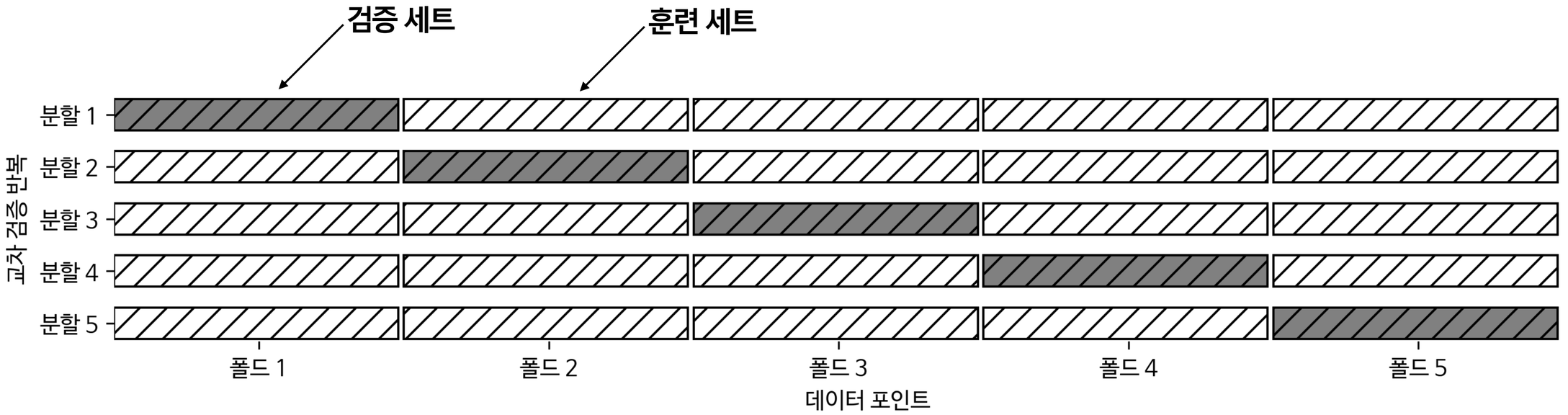
```
array([ True])
```

추론 결과. 맞았나?

성능측정: 교차 검증

- 훈련 세트 중 일부를 검증(개발) 세트로 사용하여 모델의 성능을 추정함.

- 일반화 성능을 왜곡하지 않으려고 테스트 세트를 사용하지 않음
(테스트 세트에 과대적합을 피하려고 혹은 테스트 세트의 정보 누설을 막으려고)



성능측정 : 교차검증 구현 : cross_val_score( )

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

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

cross_validate() 3개 폴드
SGD 분류기는 대략 95% 성능
인식기가 언제나 0을 출력해도 약 90%
(0인 것이 90%이므로 : 불균형 데이터셋)

```
from sklearn.model_selection import StratifiedKFold
from sklearn.base import clone
skfolds = StratifiedKFold(n_splits=3, random_state=42)
for train_index, test_index in skfolds.split(X_train, y_train_5):
    clone_clf = clone(sgd_clf)
    X_train_folds = X_train[train_index]
    y_train_folds = (y_train_5[train_index])
    X_test_fold = X_train[test_index]
    y_test_fold = (y_train_5[test_index])
    clone_clf.fit(X_train_folds, y_train_folds)
    y_pred = clone_clf.predict(X_test_fold)
    n_correct = sum(y_pred == y_test_fold)
    print(n_correct / len(y_pred)) # prints 0.9502, 0.96565 and 0.96495
```

cross_validate() 을 사용하는 대신
이렇게 구현해도 됨

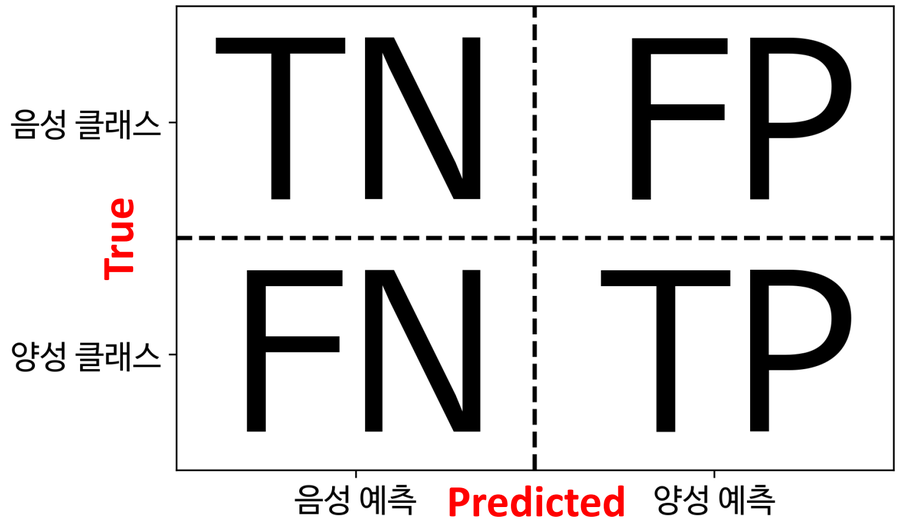
성능측정: 오차 행렬 (confusion matrix)

- 불균형 데이터셋은 정확도만으로 평가하기 어려움
- 오차행렬(confusion matrix)은 분류 모델의 성능을 평가하기 위해 사용됨

```
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]])
```



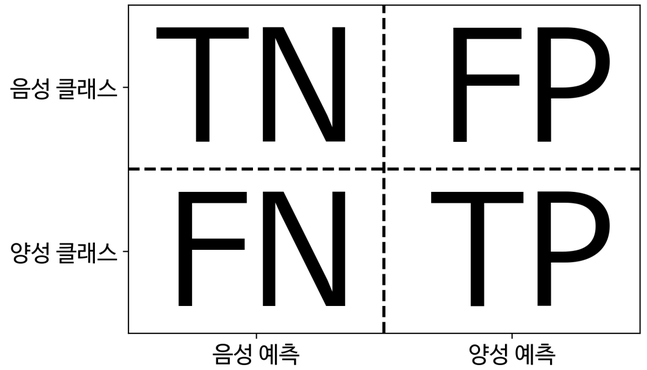
성능측정: 오차 행렬 . 정확도(accuracy)

```
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]])
```

오차행렬(confusion matrix)



정확도 (accuracy) $= \frac{TN + TP}{TN + TP + FN + FP} = \frac{53272+4344}{53272+4344+1077+1307} = \frac{57616}{60000} = 0.96$

```
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]])
```

정확도= $\frac{54579}{54579+5421} = \frac{54579}{60000} = 0.91$

오차행렬은 많이 다른데 정확도는 비슷
➔불균형 데이터셋에서 정확도는 좋은 지표가 아님

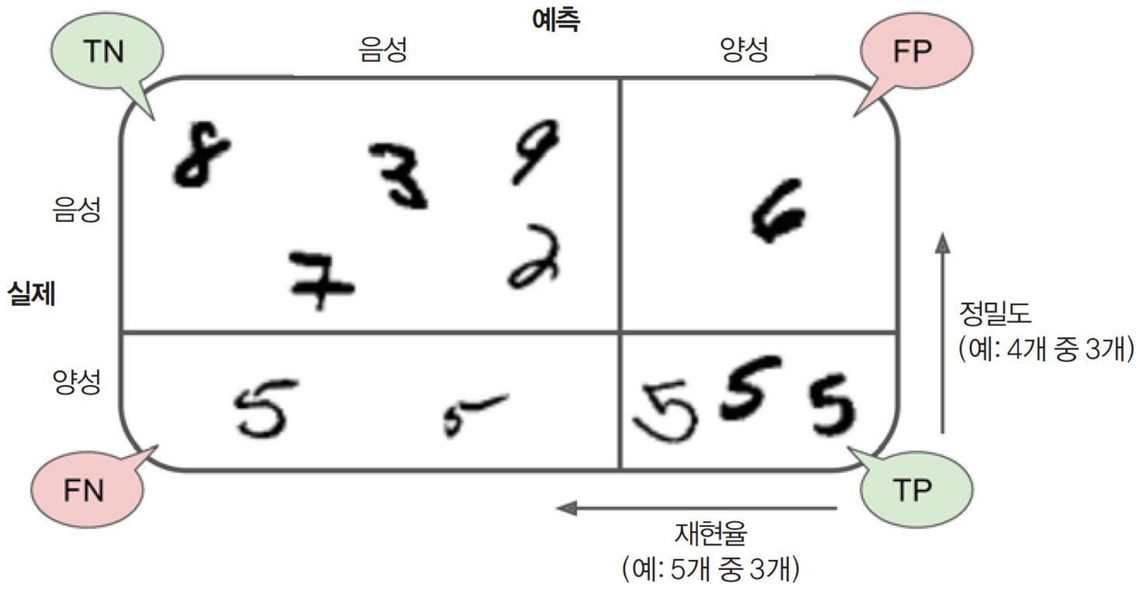
성능측정: 정밀도와 재현율

$$\text{정밀도} = \frac{TP}{TP + FP}$$

precision = 예측양성비율(PPR)
양성이라고 예측한 것 중에 정확도

$$\text{재현율} = \frac{TP}{TP + FN}$$

recall=민감도(sensitivity)=진짜양성비율(TPR)
양성인 샘플들 중에 정확도



[[53272, 1307],
[1077, 4344]]

$$\text{정밀도} = \frac{4344}{4344+1307} = \frac{4344}{5651} = 0.7687$$

$$\text{재현율} = \frac{4344}{4344+1077} = \frac{4344}{5421} = 0.8013$$

```
[[54579,     0],
 [ 5421,     0]]
```

정밀도?
재현율?
정확도?

```
from sklearn.metrics import precision_score, recall_score
precision_score(y_train_5, y_pred)     #0.7687135
recall_score(y_train_5, y_train_pred) #0.8013281
```

성능측정: f1-score, 특이도(specificity)

- f1-점수: 정밀도와 재현율의 조화 평균.

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

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

```
0.7846820809248555
```

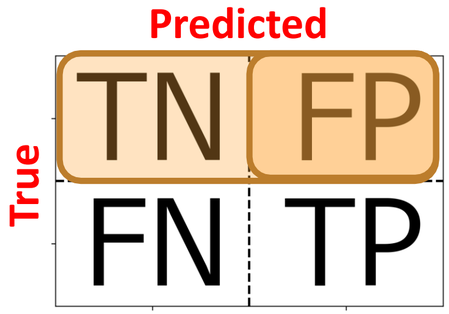
정확도는 95% 정도지만 f1-점수는 낮음

- **FPR** $= \frac{FP}{TN + FP}$

False Positive Rate
거짓양성비율

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

specificity
거짓음성비율(True Negative Rate)

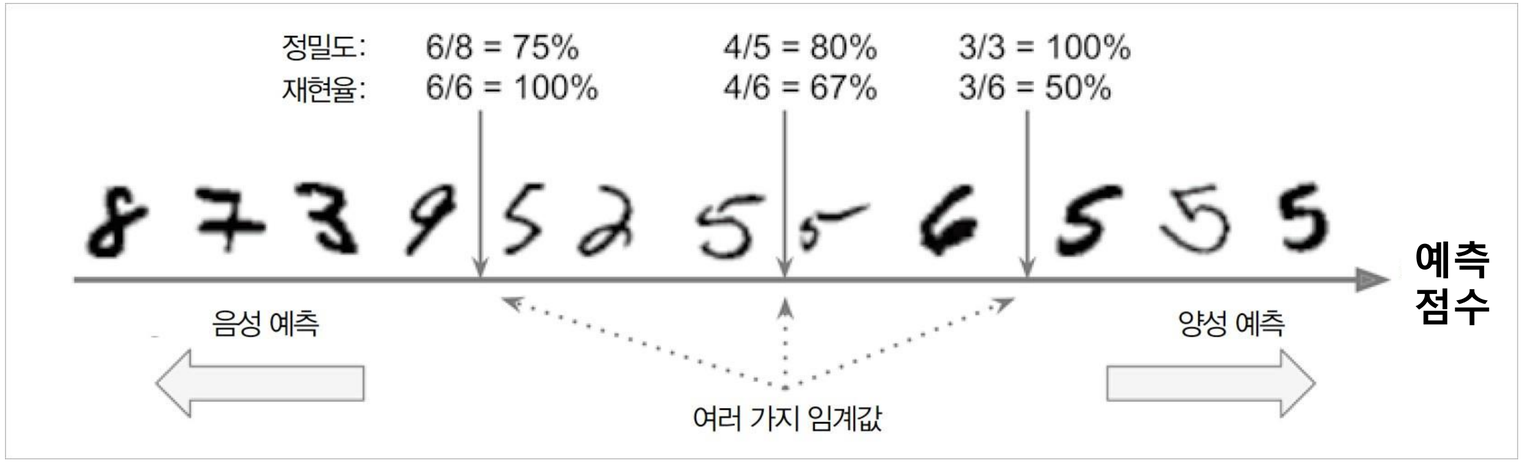


성능측정: 정밀도/재현율 트레이드오프

- 정밀도를 올리면 재현율이 줄고 그 반대도 마찬가지임.

- 예)감시카메라:
  - 재현율 높아야 됨 : 모든 도둑은 반드시 잡는다. (실제로 도둑이면 모두 검출된다)
  - 정밀도는 낮아도 됨 : 도둑이라고 해서 출동해 보았더니 도둑이 아닌 경우도 있다.

- 예)안전한 동영상 분류기:
  - 정밀도 높아야 됨 : 안전하다고 판단했으면 정말 안전한 것이다
  - 재현율 낮아도 됨 : 실제로는 안전한 것인데 안전하지 않은 것으로 분류되는 경우도 있다.



임계치를 낮춤 :
- 재현율을 높임
  더 많이 검출되므로 진짜가 많이 들어감
  (재현율이 같거나 증가함)
- 정밀도(검출된 것 중에서 진짜인 것)는
  낮아질 가능성이 커짐
  (꼭 낮아지는 것은 아님)

성능측정 : decision_function()

- decision_function() : 예측값 반환.
- 양성 예측에 대한 확신이 높을수록 decision_function()의 값이 높음

```
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 판단

```
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 판단

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

음성으로 예측

성능측정 : precision_recall_curve()

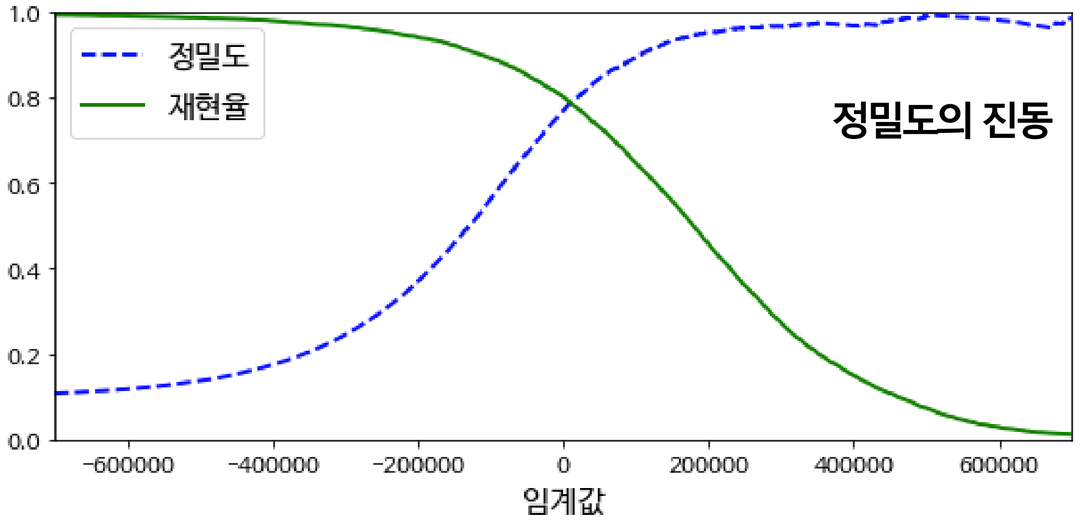
- 모든 샘플에 대한 예측값을 구함

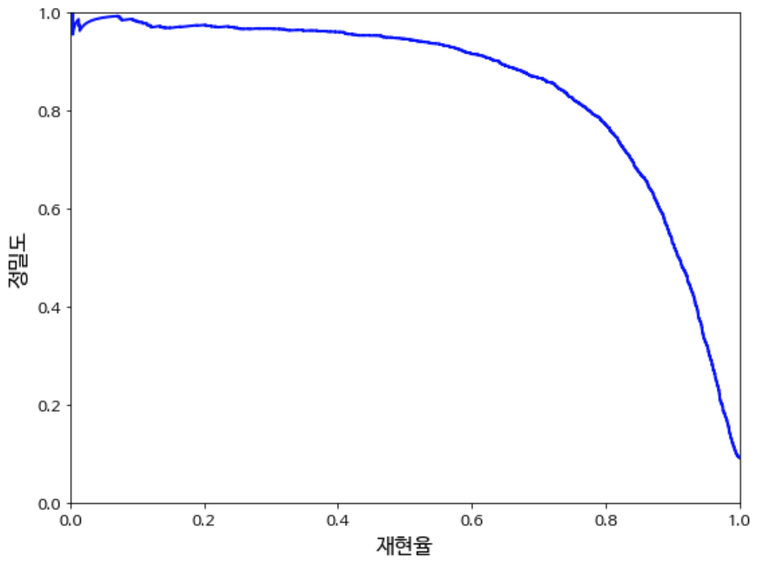
```
y_scores = cross_val_predict(sgd_clf, X_train, y_train_5, cv=3,
                             method="decision_function")
```

method='predict' 기본값,
'predict_proba'도 가능

- 임계값에 따라 정밀도와 재현율 값을 반환함 ➔ 그래프로 그릴 수 있음

```
from sklearn.metrics import precision_recall_curve

precisions, recalls, thresholds = precision_recall_curve(y_train_5, y_scores)
```

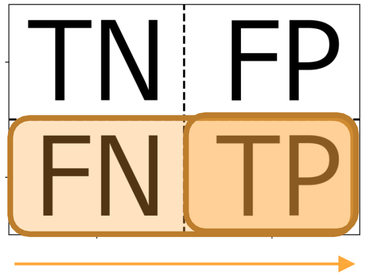




성능측정 : ROC curve

$$\text{재현율} = \frac{TP}{TP + FN}$$

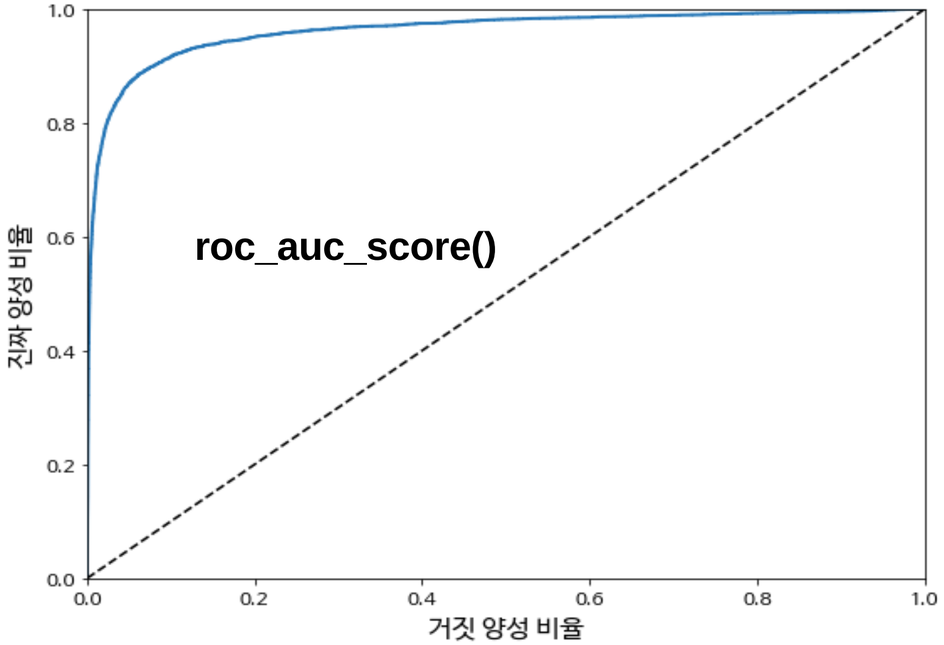
$$\text{FPR} = \frac{FP}{TN + FP}$$



- Receiver Operating Cureve(수신기 조작 곡선)
- FPR(거짓양성비율) vs. TPR(진짜양성비율, 재현 율)
- 왼꼭 위 꼭지점에 근접할수록 좋음

```
from sklearn.metrics import roc_curve

fpr, tpr, thresholds = roc_curve(y_train_5, y_scores)
```

결정함수 값



- AUC(Area Under Curve) : 클수록 좋음

```
from sklearn.metrics import roc_auc_score

roc_auc_score(y_train_5, y_scores)
```

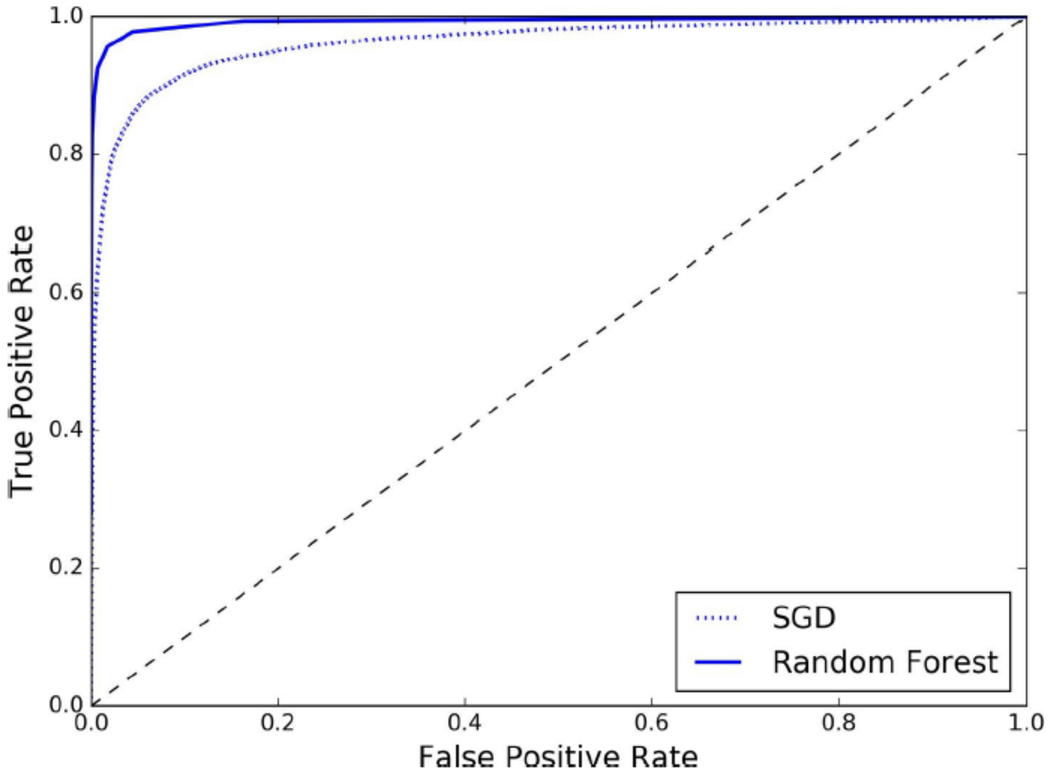
```
0.9624496555967155
```

성능측정: ROC curve 비교

```
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] # score = proba of positive class

fpr_forest, tpr_forest, thresholds_forest = roc_curve(y_train_5,y_scores_forest)
plt.plot(fpr, tpr, "b:", label="SGD")  # 앞에서 한 SGD Classifier 결과 plot
plot_roc_curve(fpr_forest, tpr_forest, "Random Forest")  # random forest 결과 plot
plt.legend(loc="bottom right")
plt.show()
```

어떤 것이 더 좋은가?

AUC는 어떤 것이 더 클까?



다중 분류

- 다중 분류 : 두 개 이상의 클래스를 구별하는 것
- 다중 분류 가능한 모델 : 결정트리, 랜덤포리스트, 나이브베이즈, 로지스틱 회기, 신경회로망, 딥러닝
- 2진 분류만 가능한 모델 : 선형분류기, SVM(Support Vector Machine)
- 2진 분류기 여러 개로 다중 분류를 구현하는 방법 : OvA, OvO

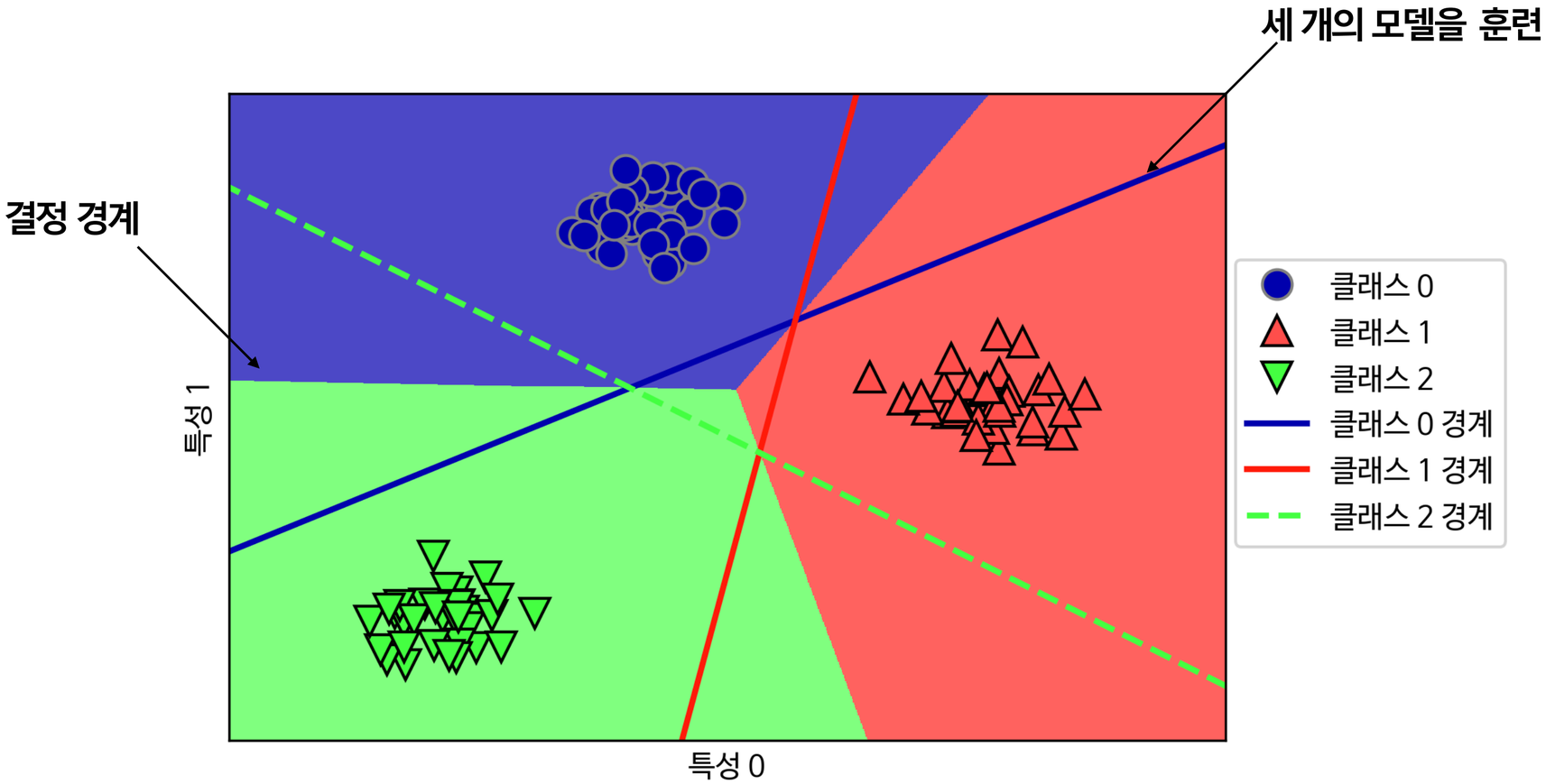
✓ OvA : One-versus-All, One-versus-Rest 클래스 하나와 나머지를 분류하는 방법 : N개 분류기

- $Output\ class = \arg\max_i C_i$, C_i : $classification\ score\ of\ class\ i\ over\ non\_i$
- 대부분 분류기의 default 값

✓ OvO : One-versus-One 두 개의 클래스 마다 분류기를 훈련하는 방법 : N x (N-1) / 2 개 분류기

- $Output\ class = \arg\max_i \sum_j C_{ij}$, C_{ij}: $classification\ result\ of\ class\ i\ \ over\ class\ j$.
  ($=$ 1 if classified to class i, 0 if classified to class j)
- SVM의 default 값

다중분류: One-versus-All



다중분류: SGDClassfier : OvA

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

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

원본 타깃 데이터 사용

```
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 계산됨
default : OvA

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

```
5
```

가장 큰 인덱스가 예측에 사용됨

```
sgd_clf.classes_
```

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

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

```
5.0
```

인덱스로부터 클래스 값.
클래스의 인덱스는 클래스 값이 아님

다중분류: 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

다중분류: RandomForest : 원래 다중분류

```
from sklearn.ensemble import RandomForestClassifier
forest_clf = RandomForestClassifier(random_state=42)
forest_clf.fit(X_train, y_train)
forest_clf.predict([some_digit])
# array([5.])
```

다중분류: 다중 분류 교차 검증

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

기본적으로 OvA. SVM은 OvO

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

정확도

```
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")
```

평균0, 분산 1로 스케일링 하니

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

인식율이 높아졌다

다중분류: 오차 행렬(Confusion Matrix)

```
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
```

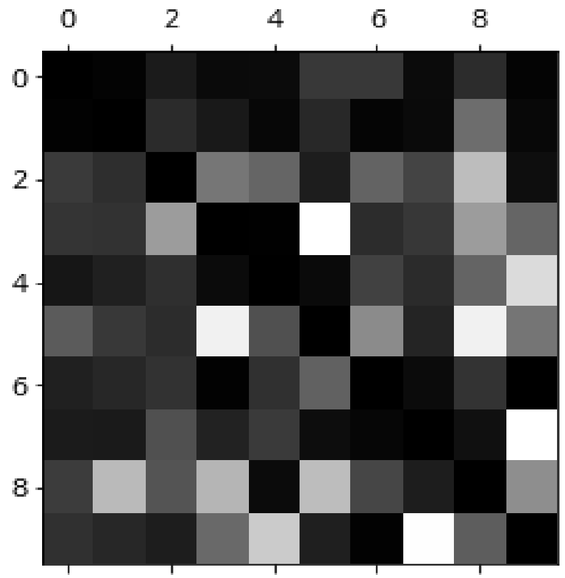
```
array([[5725,    3,   24,    9,   10,   49,   50,   10,   39,    4],
       [   2, 6493,   43,   25,    7,   40,    5,   10,  109,    8],
       [  51,   41, 5321,  104,   89,   26,   87,   60,  166,   13],
       [  47,   46,  141, 5342,    1,  231,   40,   50,  141,   92],
       [  19,   29,   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
```

오차율

```
np.fill_diagonal(norm_conf_mx, 0)
plt.matshow(norm_conf_mx, cmap=plt.cm.gray)
plt.show()
```

대각선 0으로 만들고 그림



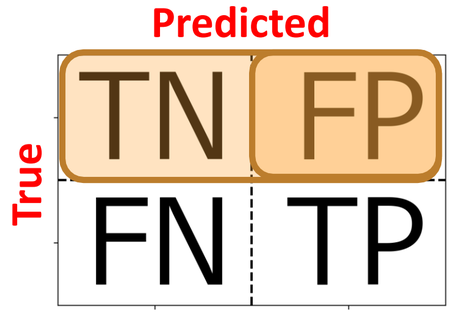
3를 5로 오인식하는 비율과
5를 3로 오인식하는 비율
어떤 것이 큰가?

다중분류: 다중 분류 리포트

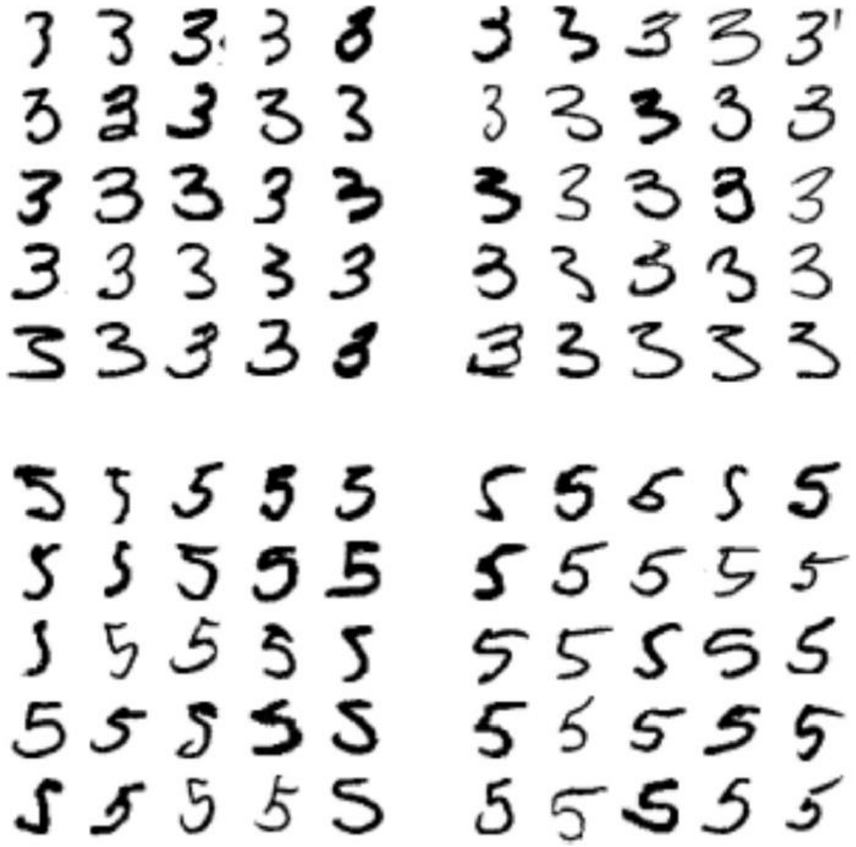
```
print(classification_report(y_train, y_train_pred))
```

```
             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
```

다중분류: 인식결과 보기




```
cl_a, cl_b = 3, 5
X_aa = X_train[(y_train == cl_a) & (y_train_pred == cl_a)]
X_ab = X_train[(y_train == cl_a) & (y_train_pred == cl_b)]
X_ba = X_train[(y_train == cl_b) & (y_train_pred == cl_a)]
X_bb = X_train[(y_train == cl_b) & (y_train_pred == cl_b)]
plt.figure(figsize=(8,8))
plt.subplot(221); plot_digits(X_aa[:25], images_per_row=5)
plt.subplot(222); plot_digits(X_ab[:25], images_per_row=5)
plt.subplot(223); plot_digits(X_ba[:25], images_per_row=5)
plt.subplot(224); plot_digits(X_bb[:25], images_per_row=5)
plt.show()
```



다중 레이블 분류(Multilabel Classification)

- 여러개의 이진 레이블을 출력(ex, 사진에 밥, 앨리스, 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)
```

DecisionTreeClassifier, RandomForestClassifier,
MLPClassifier, KNeighborsClassifier : 다중레이블 지원

두 개의 레이블:7보다 큰지와 홀수 여부

```
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]])
```

두 개의 레이블을 예측

다중 출력 분류(Multioutput Classification)

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

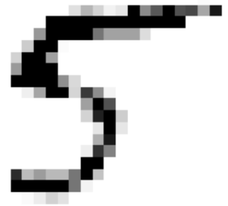
```
noise = np.random.randint(0, 100, (len(X_train), 784))
X_train_mod = X_train + noise
noise = np.random.randint(0, 100, (len(X_test), 784))
X_test_mod = X_test + noise
y_train_mod = X_train
y_test_mod = X_test
```

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