머신러닝 톺아보기 세션

3주차. 분류모델

유선호

0 CONTEXT

1. MNIST
2. 이진 분류기 훈련
3. 성능 측정
4. 다중 분류
5. 에러 분석
6. 다중 레이블 분류
7. 다중 출력 분류

1 MNIST

MNIST 데이터셋

미국 고등학생과 인구조사국 직원들이 손으로 쓴 70,000개의 숫자 이미지로 구성된 데이터셋

사용된 0부터 9까지의 숫자는 28*28 = 784 크기의 픽셀로 구성된 이미지 데이터

2차원 array가 아닌 길이가 784인 1차원 array로 제공

레이블 : 총 70,000개의 사진 샘플이 표현하는 값

WeTIE

1 MNIST

문제정의

지도학습 : 각 이미지가 담고 있는 숫자가 레이블로 지정됨.

분류 : 이미지 데이터를 분석하여 0부터 9까지의 숫자로 분류

이미지 그림을 총 10개의 클래스로 분류하는 다중 클래스 분류(multiclass classification)

배치 또는 온라인 학습 : 둘 다 가능

확률적 경사하강법(stochastic gradient descent, SGD) : 배치와 온라인 학습 모두 지원

랜덤 포레스트 분류기 : 배치 학습

1 MNIST

```
from sklearn.datasets import fetch_openml
mnist = fetch_openml('mnist_784', version = 1)
mnist.keys()
```

```
dict_keys(['data', 'target', 'frame', 'categories', 'feature_names', 'target_names', 'DESCR', 'details', 'url'])
```

```
X, y = mnist['data'], mnist['target']
print(X.shape)
print(y.shape)
```

```
(70000, 784)
(70000,)
```

```
import matplotlib.pyplot as plt

some_digit = X.iloc[0]
some_digit_image = some_digit.to_numpy().reshape(28, 28)

plt.imshow(some_digit_image, cmap = "binary")
plt.axis("off")
plt.show()
```

WeTIE

1 MNIST

```
y[0]
```

```
5
```

```
import numpy as np
y = y.astype(np.uint8)
```

WeTIE

1 MNIST

훈련 셋과 데이터 셋 나누기

MNIST 데이터셋은 이미 6:1로 분류되어 있음

훈련 세트 : 앞쪽 60,000개 이미지

테스트 세트 : 나머지 10,000개의 이미지

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

WeTIE

2 이진 분류기 훈련

예제: 5-감지기

이미지 샘플이 5를 표현하는지 여부를 판단하는 이진 분류기

이를 위해 레이블을 0 또는 1로 수정

기존에 0부터 9까지의 숫자 대신 0, 1. 두 값으로만 구성된 새로운 타깃을 생성해서 학습에 사용

숫자 5를 가리키는 이미지 레이블 : 1

숫자 5 이외의 수를 가리키는 이미지 레이블 : 0

```
y_train_5 = (y_train == 5) # 5는 True고, 다른 숫자는 모두 False
y_test_5 = (y_test == 5)
```

WeTIE

2 이진 분류기 훈련

SGDClassifier : 확률적 경사 하강법

매우 큰 데이터셋을 효율적으로 처리하는 장점을 가짐.

한 번에 하나씩 훈련 샘플을 독립적으로 처리한다는 특징을 가짐.

```
from sklearn.linear_model import SGDClassifier

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

```
SGDClassifier
SGDClassifier(random_state=42)
```

```
sgd_clf.predict([X.iloc[0]])
```

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

WeTIE

3 성능 측정

1. 교차 검증을 사용한 정확도 측정

2장에서 배운 교차검증 기술을 이용하여 SGD 분류기의 성능을 측정

성능 측정 기준 : 정확도

정확도 : 전체 샘플을 대상으로 정확하게 예측한 비율 (숫자 5를 표현하는 이미지를 True로 예측한 비율)

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

```
array([0.95035, 0.96035, 0.9604 ])
```

3 성능 측정

1. 교차 검증을 사용한 정확도 측정

교차 검증 결과가 95% 이상으로 매우 우수한 것으로 나옴.

하지만 무조건 '5 아님'이라고 찍는 분류기도 90%의 정확도를 보임.

훈련 세트의 샘플이 불균형적으로 구성되었다면, 정확도를 분류기의 성능 측정 기준으로 사용하는 것은 피해야 함

2. 오차 행렬(Confusion matrix)

오차 행렬 : 클래스 별 예측 결과를 정리한 행렬

오차 행렬의 행은 실제 클래스, 열은 예측된 클래스를 가리킴

클래스 A의 샘플이 클래스 B의 샘플로 분류된 횟수를 알고자 하면 A행 B열의 값을 확인하면 된다.

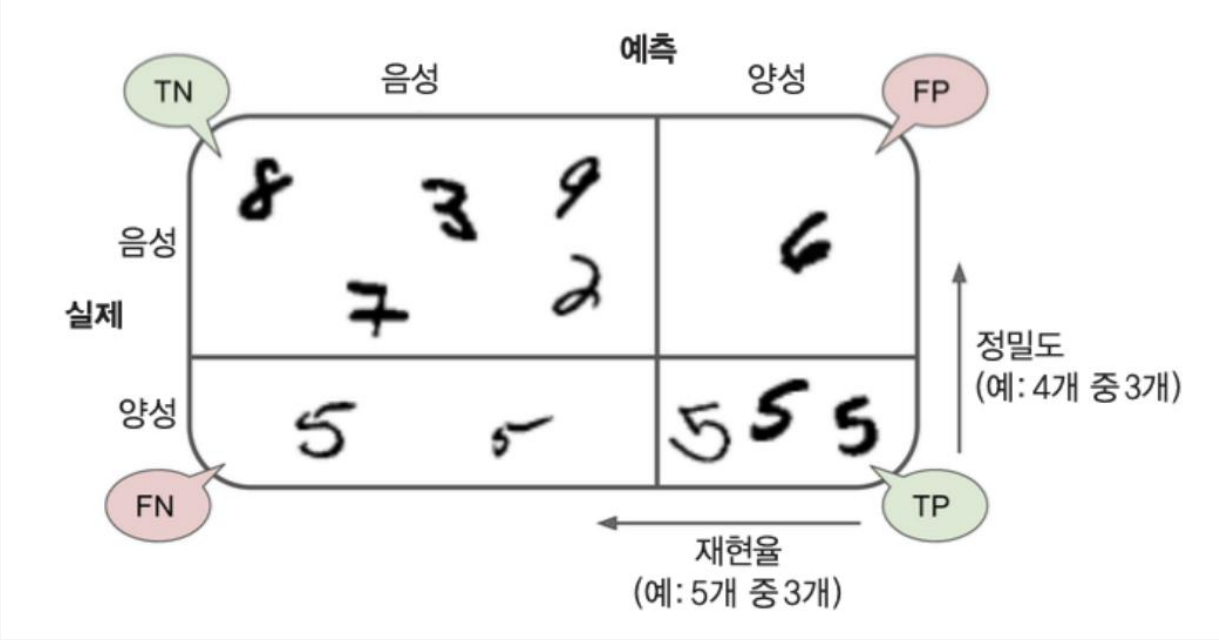
예제 : '숫자 5-감지기 ' 에 대한 오차 행렬 생성

결과는 2x2 모양의 2차원 배열

레이블의 값이 0과 1 두개의 값으로 구성되기 때문이다.

3 성능 측정

2. 오차 행렬(Confusion matrix)



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

TN(True Negative) : 실제가 Negative인데, 예측을 Negative로 한 경우

FN(False Negative) : 실제가 Positive인데, 예측을 Negative로 한 경우

FP(False Positive) : 실제가 Negative인데, 예측을 Positive로 한 경우

TP(True Positive) : 실제가 Positive인데, 예측을 Positive로 한 경우

WeTIE

2. 오차 행렬(Confusion matrix)

```
from sklearn.model_selection import cross_val_predict
from sklearn.metrics import confusion_matrix

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

```
array([[53892,   687],
       [ 1891,  3530]], dtype=int64)
```

```
y_train_perfect_predictions = y_train_5 # 완벽한 분류기일 경우
confusion_matrix(y_train_5, y_train_perfect_predictions)
```

```
array([[54579,     0],
       [    0,  5421]], dtype=int64)
```

3. 정밀도와 재현율

정밀도(precision)

양성 예측의 정확도, 숫자 5라고 예측된 값들 중에 진짜로 5인 숫자들의 비율

$$\text{precision} = \frac{TP}{TP + FP}$$

정밀도 하나만으로 분류기의 성능을 평가할 수는 없음

숫자 5를 가리키는 이미지 중에서 숫자 5가 아니라고 판명한 경우를 고려하지 않음

분류기가 정확하게 예측한 양성 샘플의 비율이 재현율을 함께 다루어야 함

3. 정밀도와 재현율

재현율(recall)

양성 샘플에 대한 정확도, 분류기가 정확하게 감지한 양성 샘플의 비율

민감도(sensitivity), 참 양성 비율(true positive rate)로도 부름

$$\text{recall} = \frac{TP}{TP + FN}$$

F1 점수(F1 score)

정밀도와 재현율의 조화 평균

$$F_1 = \frac{2}{\frac{1}{precision} + \frac{1}{recall}} = \frac{2 \cdot precision \cdot recall}{precision + recall}$$

3. 정밀도와 재현율

F1 점수가 높을수록 분류기의 성능을 좋게 평가하지만, 경우에 따라 조심할 필요가 있음

경우에 따라 재현율과 정밀도 둘 중의 하나에 높은 가중치를 두어야 할 때가 있다.

앞서 정의된 F1 점수는 재현율과 정밀도의 중요도가 동일하다고 가장한 결과

예를 들어, 암 진단 결과와 관련해서 아래 두 경우에 발생하는 비용이 다름

실제로 음성이지만, 양성으로 오진(정밀도)

실제로 양성이지만 음성으로 오진(재현율)

```
from sklearn.metrics import precision_score, recall_score

print(precision_score(y_train_5, y_train_pred)) # 3530 / (3530 + 687)
print(recall_score(y_train_5, y_train_pred)) # 3530 / (3530 + 1891)
```

```
0.8370879772350012
0.6511713705958311
```

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

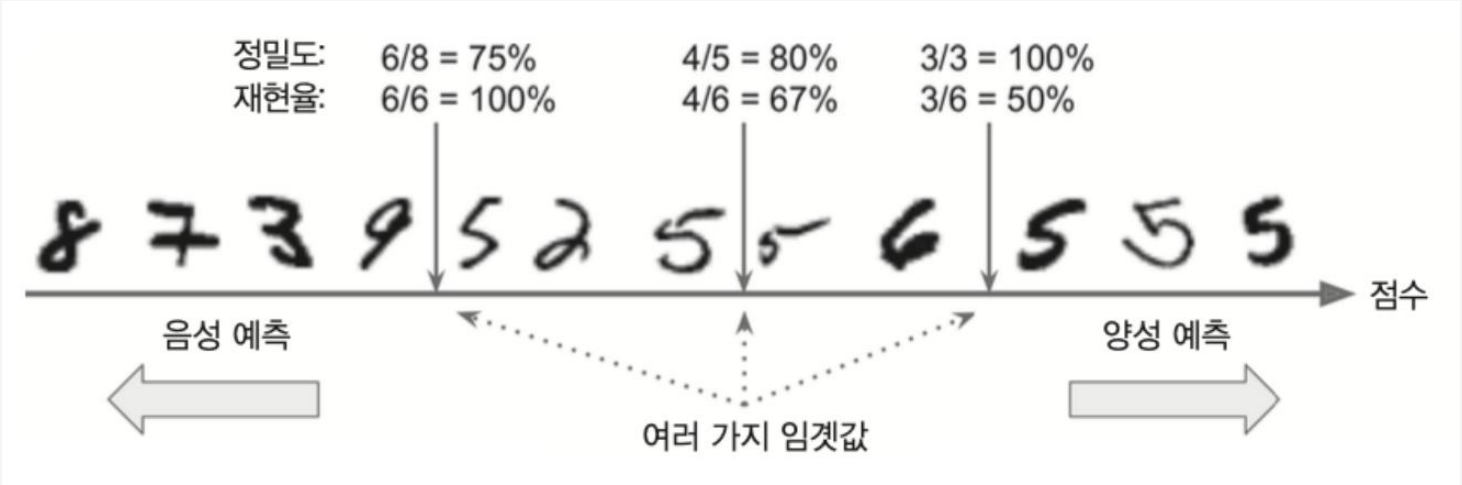
```
0.7325171197343847
```

4. 정밀도/재현율 트레이드오프

결정 함수와 결정 임계값

결정 함수(decision function) : 분류기가 각 샘플의 점수를 계산할 때 사용

결정 임계값(decision threshold) : 결정 함수의 값이 이 값보다 같거나 크면 양성 클래스로 분류, 아니면 음성 클래스



4. 정밀도/재현율 트레이드오프

결정 함수와 결정 임계값

```
y_scores = sgd_clf.decision_function([X.iloc[0]])
y_scores
```

```
array([2164.22030239])
```

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

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

```
threshold = 8000
y_some_digit_pred = (y_scores > threshold)
y_some_digit_pred
```

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

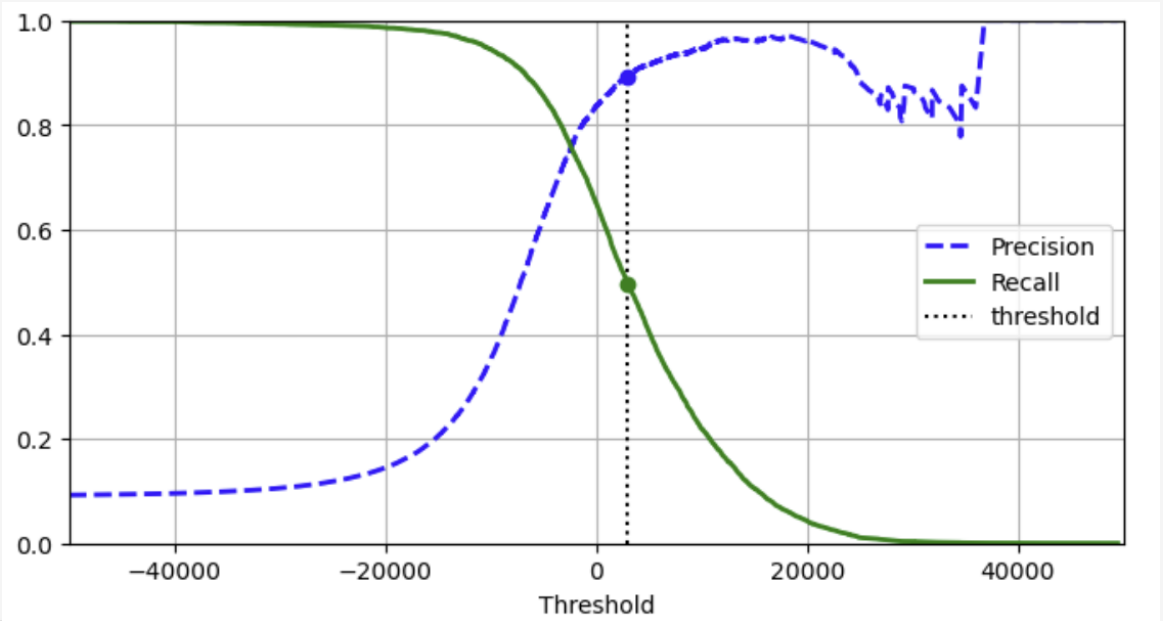
```
y_scores = cross_val_predict(sgd_clf, X_train, y_train_5, cv=3, method = "decision_function")
```

4. 정밀도/재현율 트레이드오프

임계값이 커질수록

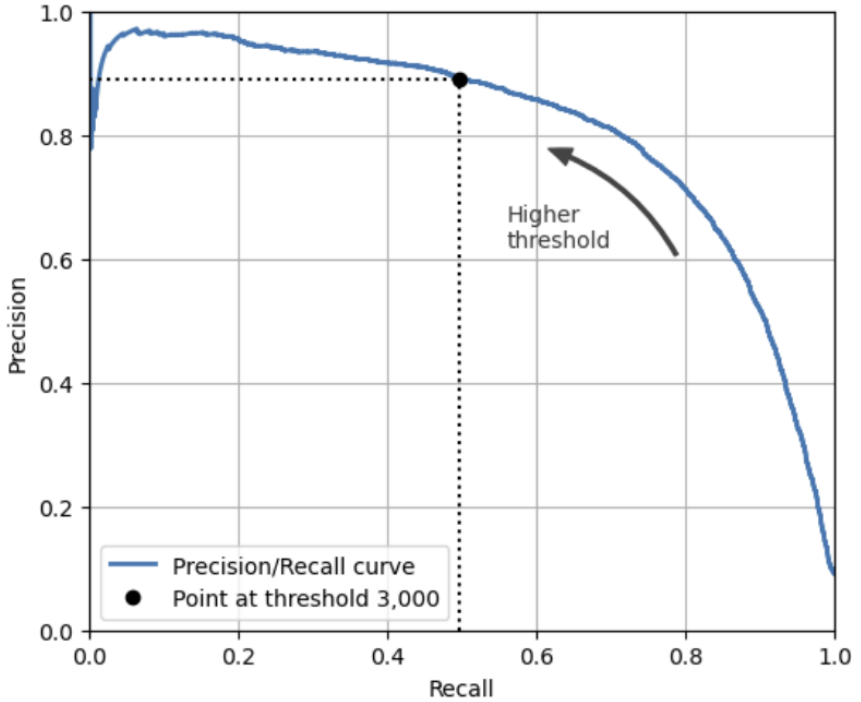
정밀도 올라가고, 재현율은 떨어짐

```
plt.figure(figsize=(8, 4))  # 그래프 크기 지정

# 정밀도 그래프
plt.plot(thresholds, precisions[:-1], "b--", label="Precision", linewidth=2)
# 재현율 그래프
plt.plot(thresholds, recalls[:-1], "g-", label="Recall", linewidth=2)

# 결정 임계값이 3000인 경우 확인
threshold = 3000
plt.vlines(threshold, 0, 1.0, "k", "dotted", label="threshold")

# 결정 임계값이 3000일 때의 정밀도와 재현율 표시
idx = (thresholds >= threshold).argmax()
plt.plot(thresholds[idx], precisions[idx], "bo")
plt.plot(thresholds[idx], recalls[idx], "go")

plt.axis([-50000, 50000, 0, 1])  # x축 구간
plt.grid()                       # 그리드 그리기
plt.xlabel("Threshold")          # x축 라벨
plt.legend(loc="center right")   # 범례 위치치

plt.show()
```



4. 정밀도/재현율 트레이드오프

```
import matplotlib.patches as patches  # 화어진 화살표 그리기 용도

plt.figure(figsize=(6, 5))  # extra code – not needed, just formatting

plt.plot(recalls, precisions, linewidth=2, label="Precision/Recall curve")

# 결정 임계값이 3,000일 때의 정밀도, 재현율을 표시
plt.plot([recalls[idx], recalls[idx]], [0., precisions[idx]], "k:")
plt.plot([0.0, recalls[idx]], [precisions[idx], precisions[idx]], "k:")
plt.plot([recalls[idx]], [precisions[idx]], "ko",
         label="Point at threshold 3,000")

# 화어진 화살표
plt.gca().add_patch(patches.FancyArrowPatch(
    (0.79, 0.60), (0.61, 0.78),
    connectionstyle="arc3,rad=.2",
    arrowstyle="Simple, tail_width=1.5, head_width=8, head_length=10",
    color="#444444"))

# 화어진 화살표 아내 문구 입력력
plt.text(0.56, 0.62, "Higher\nthreshold", color="#333333")

plt.xlabel("Recall")
plt.ylabel("Precision")
plt.axis([0, 1, 0, 1])
plt.grid()
plt.legend(loc="lower left")

plt.show()
```




```
threshold_90_precision = thresholds[np.argmax(precisions >= 0.90)]
y_train_pred_90 = (y_scores >= threshold_90_precision)

print(precision_score(y_train_5, y_train_pred_90))
print(recall_score(y_train_5, y_train_pred_90))
```

```
0.9000345901072293
0.4799852425751706
```

WeTIE

5. ROC 곡선

수신기 조작 특성(receiver operating characteristic, ROC) 곡선을 활용하여 이진 분류기의 성능 측정 가능

결정 임계값에 따른 거짓 양성 비율(false positive rate, FPR)에 대한 참 양성 비율(true positive rate, TPR)의 관계를 나타낸 곡선

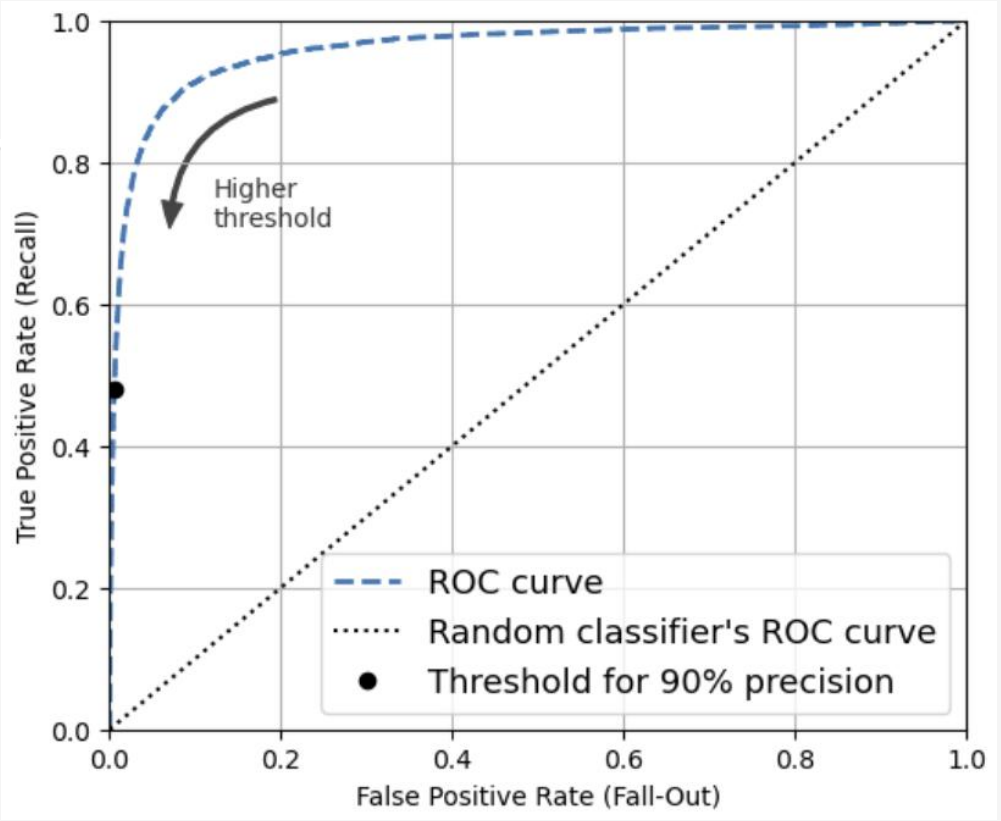
참 양성 비율 : 재현율

거짓 양성 비율 : 원래 음성인 샘플 중에서 양성이라고 잘못 분류된 샘플들의 비율

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

5. ROC 곡선

```
from sklearn.metrics import roc_curve

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

plt.figure(figsize=(6, 5))

# ROC 커브 그리기
plt.plot(fpr, tpr, '--', linewidth=2, label="ROC curve")
plt.plot([0, 1], [0, 1], 'k:', label="Random classifier's ROC curve")

# 정밀도 90%의 위치 표시
idx_for_threshold_at_90 = (thresholds <= threshold_90_precision).argmax()
tpr_90, fpr_90 = tpr[idx_for_threshold_at_90], fpr[idx_for_threshold_at_90]
plt.plot([fpr_90], [tpr_90], "ko", label="Threshold for 90% precision")

# 기타 요소 표시
plt.gca().add_patch(patches.FancyArrowPatch(
    (0.20, 0.89), (0.07, 0.70),
    connectionstyle="arc3,rad=.4",
    arrowstyle="Simple, tail_width=1.5, head_width=8, head_length=10",
    color="#444444"))
plt.text(0.12, 0.71, "Higher\nthreshold", color="#333333")
plt.xlabel('False Positive Rate (Fall-Out)')
plt.ylabel('True Positive Rate (Recall)')
plt.grid()
plt.axis([0, 1, 0, 1])
plt.legend(loc="lower right", fontsize=13)

plt.show()
```




```
from sklearn.metrics import roc_auc_score

roc_auc_score(y_train_5, y_scores)
```

```
0.9604938554008616
```

WeTIE

5. ROC 곡선

AUC와 분류기 성능

재현율(TPR)과 거짓 양성 비율(FPR) 사이에도 서로 상쇄하는 기능이 있다는 것을 확인 가능

재현율(TPR)을 높이고자 하면 거짓 양성 비율(FPR)도 함께 증가

따라서 좋은 분류기는 재현율은 높으면서 거짓 양성 비율은 최대한 낮게 유지해야 함

ROC 곡선이 y축에 최대한 근접하는 결과가 나오도록 해야함

AUC : ROC 곡선 아래의 면적

이 면적이 1에 가까울수록 성능이 좋은 분류기로 평가

5. ROC 곡선

SGD와 랜덤 포레스트의 AUC 비교

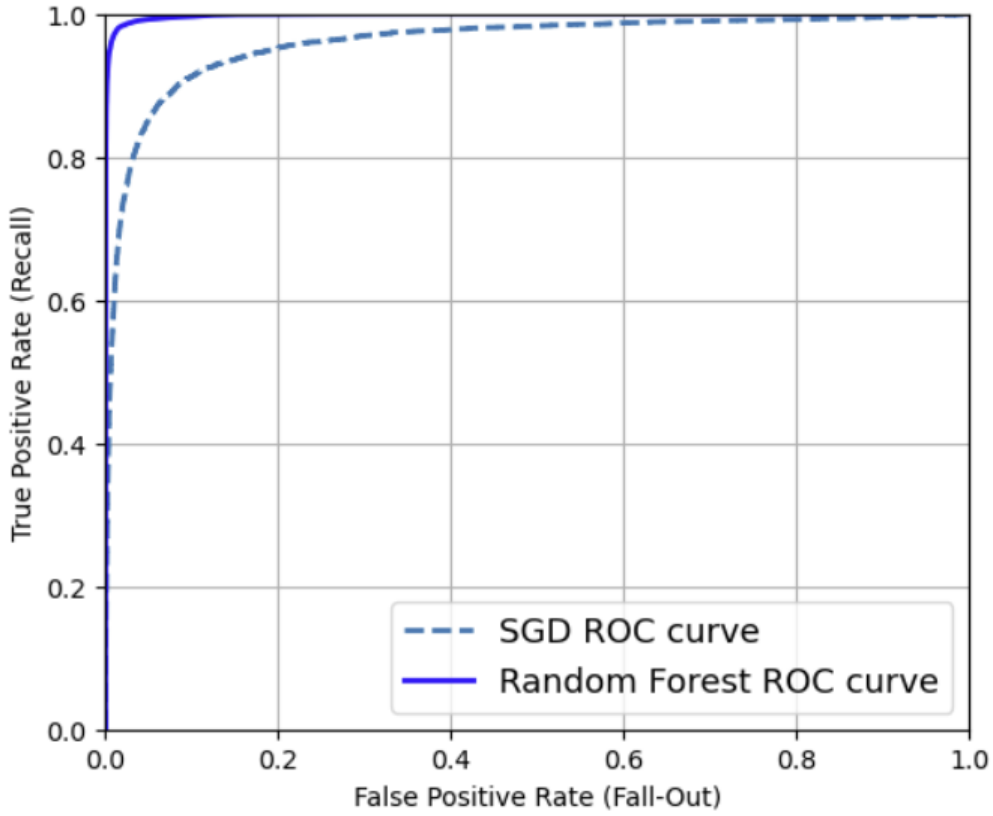
```
from sklearn.ensemble import RandomForestClassifier

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]  # 양성일 확률

# 임계값에 따른 정밀도, 재현율 계산
precisions_forest, recalls_forest, thresholds_forest = precision_recall_curve(
    y_train_5, y_scores_forest)
```

5. ROC 곡선

SGD와 랜덤 포레스트의 AUC 비교

```
fpr_forest, tpr_forest, thresholds_forest = roc_curve(y_train_5, y_scores_forest)

plt.figure(figsize=(6, 5))

# ROC 커브 그리기
plt.plot(fpr, tpr, '--', linewidth=2, label="SGD ROC curve")
plt.plot(fpr_forest, tpr_forest, 'b-', linewidth=2, label="Random Forest ROC curve")

plt.xlabel('False Positive Rate (Fall-Out)')
plt.ylabel('True Positive Rate (Recall)')
plt.grid()
plt.axis([0, 1, 0, 1])
plt.legend(loc="lower right", fontsize=13)

plt.show()
```




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

```
0.9983436731328145
```

WeTIE

4 다중 분류

다중 클래스 분류기(multiclass classifier)

세 개 이상의 클래스로 샘플을 분류하는 예측기

다항 분류기(multinomial classifier)라고도 부름

손글씨 숫자 분류의 경우 0부터 9까지 10개의 클래스로 분류해야 함

다중 클래스 분류 지원 분류기

- SGD 분류기
- 랜덤 포레스트 분류기
- 나이브 베이즈(naïve Bayes) 분류기

이진 분류만 지원하는 분류기

- 로지스틱 회귀
- 서포트 벡터 머신

4 다중 분류

이진 분류기 활용

이진 분류기를 활용하여 다중 클래스 분류 가능

- 일대다(OvR 또는 OvA)
- 일대일(OvO)

일대다 방식(OvA 또는 OvR)

- OvA(One-versus-All) 또는 OvR(One-versus-Rest)
- 숫자 5 예측하기에서 사용했던 이진 분류 방식을 동일하게 모든 숫자에 대해서 실행
- 각 샘플에 대해 총 10번 각기 다른 이진 분류기를 실행
- 이후 각 분류기의 결정 점수 중에서 가장 높은 점수를 받은 클래스를 선택

일대일 방식(OvO)

- One-versus-One
- 조합 가능한 모든 일대일 분류 방식을 진행하여 가장 많은 결투를 이긴 숫자를 선택
- MNIST의 경우, 9+8+…+1 = 45개의 결투를 판별하는 분류기를 이용
    - 훈련 세트의 각 샘플에 대해 총 45번 결투가 벌어지며 그 중에서 가장 높은 점수를 얻는 숫자가 선택됨

4 다중 분류

OvO 활용

서포트 벡터 머신(SVC)

대부분의 이진 분류기는 일대다 전략 사용

```
from sklearn.svm import SVC

svm_clf = SVC()
svm_clf.fit(X_train, y_train) # y_train_5가 아닌 y_train 사용
svm_clf.predict([X.iloc[0]])
```

```
array([5], dtype=uint8)
```

```
some_digit_scores = svm_clf.decision_function([X.iloc[0]])
some_digit_scores
```

```
array([[ 1.72501977,  2.72809088,  7.2510018 ,  8.3076379 , -0.31087254,
         9.3132482 ,  1.70975103,  2.76765202,  6.23049537,  4.84771048]])
```

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

```
5
```

WeTIE

4 다중 분류

OvR 활용

이진 분류기를 일대일 전략 또는 일대다 전략으로 지정해서 학습하도록 만들 수 있음.

사이킷런의 경우 : OneVsOneClassifier 또는 OneVsRestClassifier 사용

```
from sklearn.multiclass import OneVsRestClassifier

ovr_clf = OneVsRestClassifier(SVC())
ovr_clf.fit(X_train, y_train)
ovr_clf.predict([some_digit])
```

```
array([5], dtype=uint8)
```

4 다중 분류

다중 클래스 지원 분류기

SGD 분류기는 다중 클래스 분류를 직접 지원

사이킷런의 OvR, OvO 등을 적용할 필요 없음

```
sgd_clf.fit(X_train, y_train)
sgd_clf.predict([X.iloc[0]])
```

```
array([3], dtype=uint8)
```

```
sgd_clf.decision_function([X.iloc[0]])
```

```
array([[-31893.03095419, -34419.69069632,  -9530.63950739,
         1823.73154031, -22320.14822878,  -1385.80478895,
       -26188.91070951, -16147.51323997,  -4604.35491274,
       -12050.767298  ]])
```

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

```
array([0.87365, 0.85835, 0.8689 ])
```

```
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")
```

```
array([0.8983, 0.891 , 0.9018])
```

다중 클래스 분류기의 성능 평가는 교차검증을 이용하여 정확도 측정

MNIST의 경우 0부터 9까지의 숫자가 균형 있게 분포되어 있어 데이터 불균형의 문제가 발생하지 않음.

5 에러 분석

오차 행렬 활용

손글씨 클래스 분류 모델의 오차 행렬을 이미지로 표현 가능

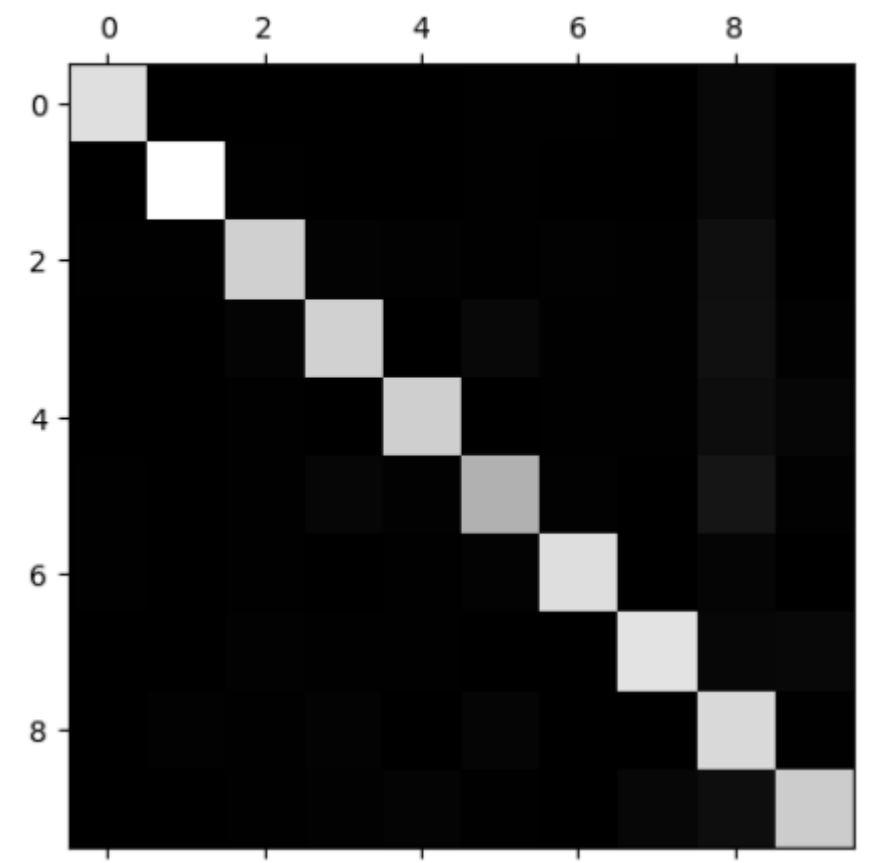
대체로 잘 분류됨 : 대각선이 밝음

5행은 좀 어두움 : 숫자 5의 분류 정확도가 상대적으로 낮음

```
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([[5577,    0,   22,    5,    8,   43,   36,    6,  225,    1],
       [   0, 6400,   37,   24,    4,   44,    4,    7,  212,   10],
       [  27,   27, 5220,   92,   73,   27,   67,   36,  378,   11],
       [  22,   17,  117, 5227,    2,  203,   27,   40,  403,   73],
       [  12,   14,   41,    9, 5182,   12,   34,   27,  347,  164],
       [  27,   15,   30,  168,   53, 4444,   75,   14,  535,   60],
       [  30,   15,   42,    3,   44,   97, 5552,    3,  131,    1],
       [  21,   10,   51,   30,   49,   12,    3, 5684,  195,  210],
       [  17,   63,   48,   86,    3,  126,   25,   10, 5429,   44],
       [  25,   18,   30,   64,  118,   36,    1,  179,  371, 5107]],
      dtype=int64)
```

```
plt.matshow(conf_mx, cmap=plt.cm.gray)
plt.show()
```



WeTIE

5 에러 분석

오차율 이미지

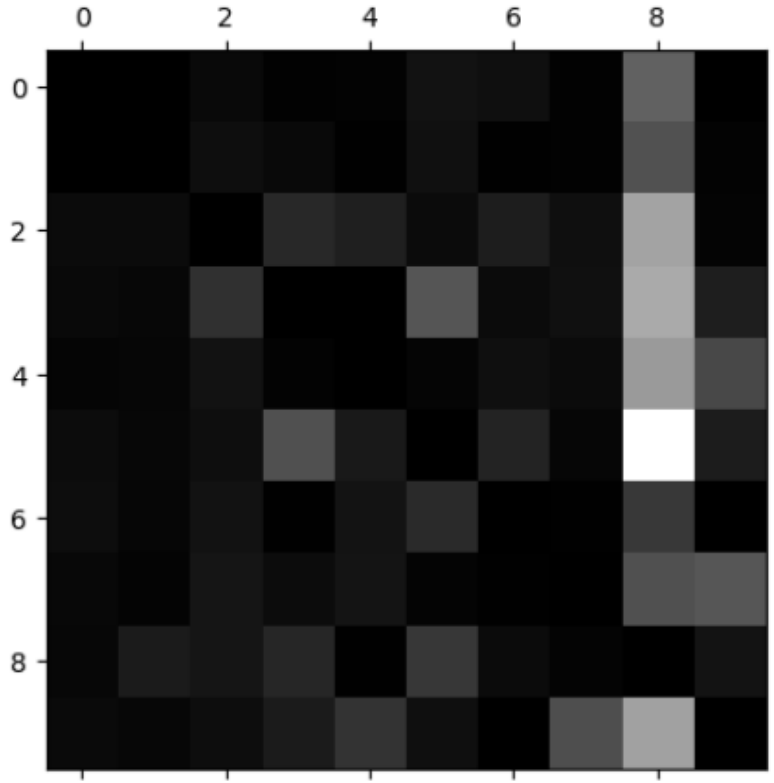
8행이 전반적으로 어두움.

8은 잘 분류되었다는 의미임.

(3, 5)와 (5, 3)의 위치가 상대적으로 밝음

3과 5가 서로 많이 혼동됨.

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

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



WeTIE

5 에러 분석

3과 5의 오차행렬 그려보기

음성 : 3으로 판정, 양성 : 5로 판정

3과 5의 구분이 어려운 이유

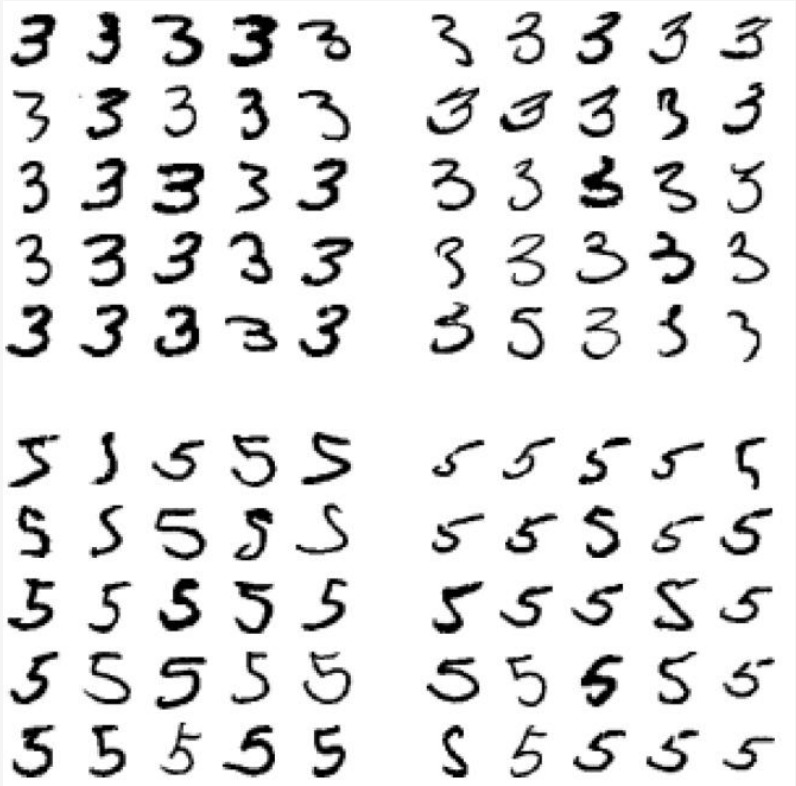
선형 모델이 SGD 분류기를 사용했기 때문

픽셀마다 가중치를 적용하여 단순히 픽셀 강도에만 의존함

이미지 분류기의 한계

이미지의 위치나 회전 방향에 민감함

이미지를 중앙에 위치시키고 회전되지 않도록 전처리하면 좀 더 좋은 성능 가능



WeTIE

6 다중 레이블 분류

샘플마다 여러 개의 클래스 출력

예제 : 얼굴 인식 분류기

한 사진에 여러 사람이 포함된 경우, 인식된 사람마다 하나씩 꼬리표(tag)를 붙여야 함.

엘리스, 밥, 찰리의 포함 여부를 확인할 때, 밥이 없는 경우: [True, False, True] 출력

예제 : 숫자 분류

7 이상인지 여부와 함께 홀수 여부도 동시 출력

5가 입력될 때 : [False, True] 출력

6 다중 레이블 분류

다중 레이블 분류 지원 모델

k-최근접 이웃 분류기

사이킷런의 KNeighborsClassifier

다중 레이블 분류기를 평가하는 방법은 다양함

모든 레이블의 가중치가 같다고 가정 : 각 레이블의 F1 점수를 구하고 평균 점수를 계산

가중치 : 레이블에 클래스의 지지도(타깃 레이블에 속한 샘플 수)를 가중치로 사용 가능

6 다중 레이블 분류

다중 레이블 분류 지원 모델

k-최근접 이웃 분류기

```
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
KNeighborsClassifier()
```

```
knn_clf.predict([X.iloc[0]])
```

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

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

```
0.9764102655606048
```

7 다중 출력 분류

다중 출력 다중 클래스 분류라고 불림

다중 레이블 분류에서 한 레이블이 다중 클래스가 될 수 있도록 일반화한 것

예제 : 이미지에서 잡음을 제거하는 시스템

잡음이 많은 숫자 이미지를 입력으로 받고, 깨끗한 숫자 이미지를 MNIST 이미지처럼 픽셀의 강도를 담은 배열로 출력

다중 레이블 : 각각의 픽셀이 레이블 역할 수행

다중 클래스 : 레이블이 0부터 255까지 픽셀 강도를 가짐

분류기를 훈련시켜 잡음이 섞인 입력 이미지를 깨끗한 타깃 이미지로 만들기

7 다중 출력 분류

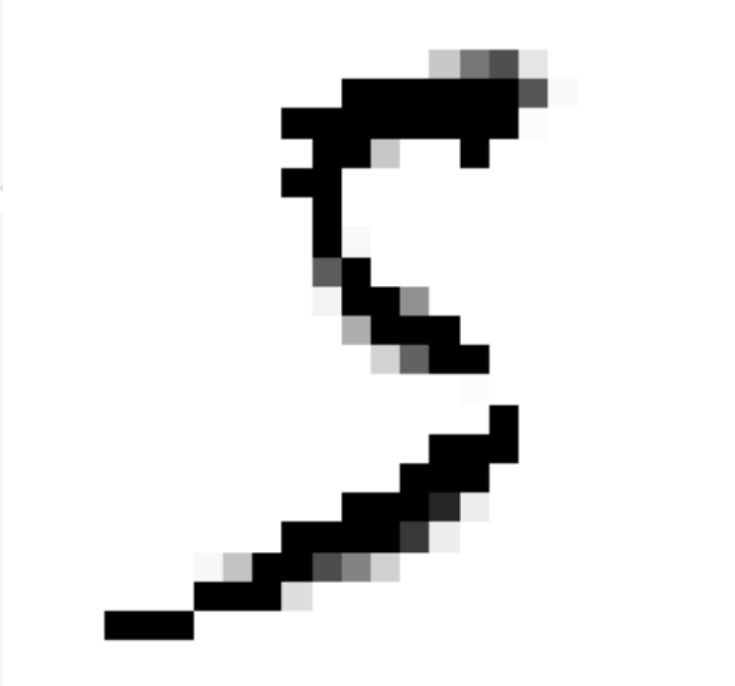
```
noise = np.random.randint(0, 100, (len(X_train), 784))
X_train_mod = X_train + noise
noise = np.random.randint(0, 100, (len(X_test), 784))
X_test_mod = X_test + noise

y_train_mod = X_train
y_test_mod = X_test
```

```
plot_digit(X_train[0])
plot_digit(X_train_mod[0])
```

WeTIE

7 다중 출력 분류

```
knn_clf.fit(X_train_mod, y_train_mod)
clean_digit = knn_clf.predict([X_train_mod[0]])

def plot_digit(image):
    image_reshaped = image.reshape(28, 28)
    plt.imshow(image_reshaped, cmap="binary")
    plt.axis("off")
    plt.show()

plot_digit(clean_digit)
```



WeTIE

Q.E.D

3주차. 분류모델