🦎 핸즈온 3장_분류

3-1. MNIST

- 고등학생과 미국 인구조사국 직원들이 손으로 쓴 70000개의 작은 숫자 이미지를 모은 데이터셋

  - 각 이미지에는 어떤 숫자를 나타내는지 레이블 되어 O
  - 70000개의 이미지, 784개의 특성
  - 28 by 28의 이미지 데이터
  - 개개의 특성은 0(흰색) ~ 255(검은색)까지의 픽셀 강도를 나타냄
- scikit-learn 내 데이터 셋의 일반적인 구조
  - 데이터셋을 설명하는 `DESCR` 키
  - 샘플이 하나의 행, 특성이 하나의 열로 구성된 배열을 가진 `data` 키
  - 레이블 배열을 담은 `target` 키
- 데이터에 대한 조사를 진행하기 전 항상 `train_test_split`을 먼저 진행해야 함
- 훈련 세트의 경우 모든 교차 검증 폴드가 비슷해야 함
  - 하나의 fold라도 특정 숫자가 누락되면 x
  - 어떤 학습 알고리즘은 훈련 sample의 순서에 민감하게 반응

  ⇒ 데이터셋을 섞어서 이러한 문제를 해결

3-2. 이진 분류기 훈련

- 이진 분류기(binary classifier)

    - yes/no만 판별
- 이진 분류 알고리즘
    - 확률적 경사 하강법(Stochastic Gradient Descent, SGD)
        - 매우 큰 데이터셋을 효율적으로 처리할 수 있는 알고리즘

          → 한 번에 하나씩 훈련 샘플을 독립적으로 처리
        - 훈련 시 무작위성을 사용(→ 확률적)
        - `sklearn.SGDClassifier`

3-3. 성능 측정

3-3-1. 교차 검증을 사용한 정확도 측정

- Stratified K-Fold
    - 클래스 별 비율이 유지되도록 fold 생성 시 계층적 샘플링을 수행하는 방법
    - 매 반복에서 분류기 객체를 복제하여 훈련 fold로 훈련시키고 테스트 fold로 예측 수행
    - 이후 올바른 예측의 수를 세어 정확한 예측의 비율을 출력
- 정확도의 함정
    - MNIST 데이터셋의 경우 이미지의 10% 정도만 숫자 5이기 때문에 무조건 '5 아님'으로 예측하면 정확도가 90%로 평가됨
    - 불균형한 데이터셋을 다룰 때(= 어떤 클래스가 다른 것보다 월등히 많은 경우) 정확도를 분류기의 성능 측정 지표로 사용하는 것을 주의해야 함

3-3-2. 오차 행렬(Confusion Matrix)

- 클래스 A의 샘플이 클래스 B로 분류된 횟수를 세는 것
- 코드

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

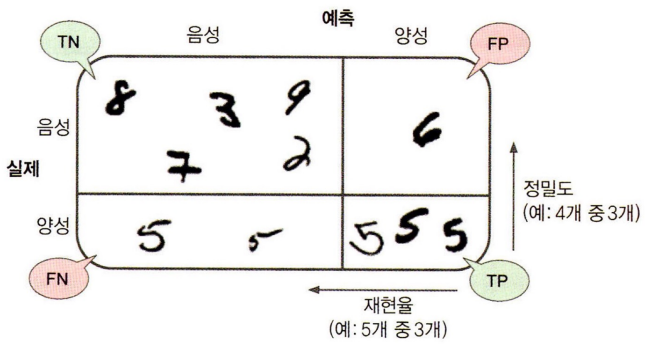
- 해석
    - 행: 실제 클래스, 열: 예측한 클래스

	예측 False	예측 True

실제 False	TN	FP
실제 True	FN	TP

  - 완벽한 분류기인 경우 오차 행렬의 주대각선만 0이 아닌 값이 됨

- 오차 행렬을 통한 요약 지표
  - 정밀도(precision)
    - 양성 예측의 정확도
    - 정밀도 = $\frac{TP}{TP+FP}$
  - 재현율(= 민감도/ TPR, recall)
    - 분류기가 정확하게 감지한 양성 샘플의 비율
    - 재현율 = $\frac{TP}{TP+FN}$



3-3-3. 정밀도와 재현율

- 사이킷런 내에 함수로 구현되어 있음
  - `sklearn.metrics.precision_score` : 정밀도
  - `sklearn.metrics.recall_score` : 재현율
- F1 score
  - 정밀도와 재현율의 조화 평균
  - $\frac{2}{\frac{1}{\text{정밀도}}+\frac{1}{\text{재현율}}} = 2 \times \frac{\text{정밀도} \times \text{재현율}}{\text{정밀도}+\text{재현율}} = \frac{TP}{TP+\frac{FN+FP}{2}}$
  - `sklearn.metrics.f1_score`
  - 정밀도와 재현율이 비슷한 경우 F1 score가 높지만, 이러한 것이 항상 바람직한 것은 아님

3-3-4. 정밀도/재현율 트레이드오프

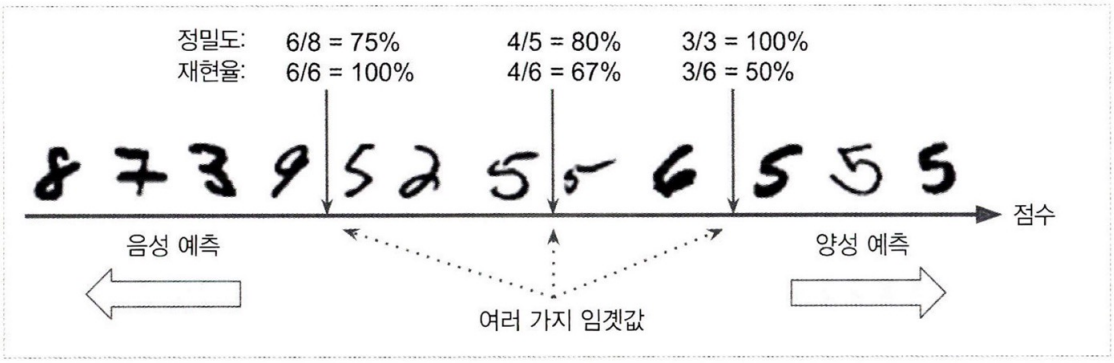
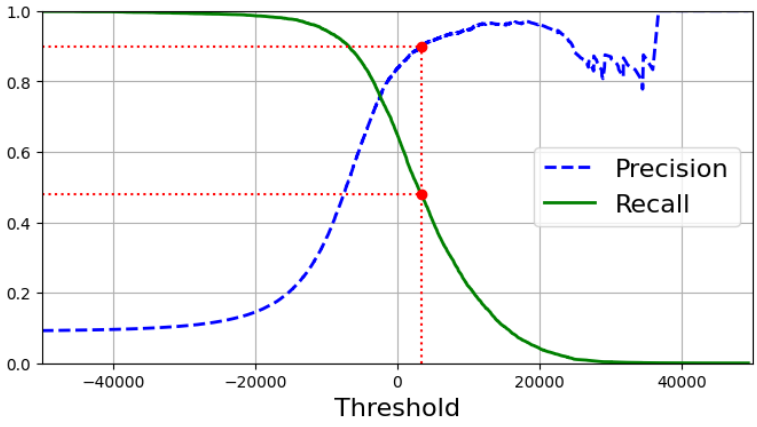
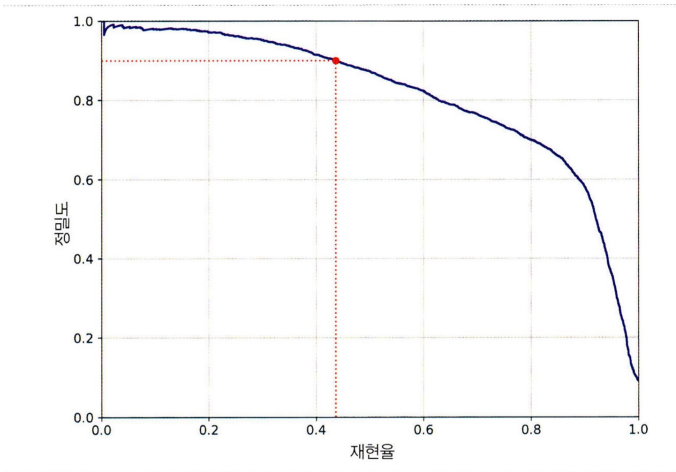


그림 3-3 이 정밀도/재현율 트레이드오프 이미지는 분류기가 만든 점수 순으로 나열되어 있습니다. 선택한 결정 임곗값 위의 것을 양성으로 판단합니다. 임곗값이 높을수록 재현율은 낮아지고 반대로 (보통) 정밀도는 높아집니다.

- 임곗값을 높이면 재현율이 줄어들고, 임곗값을 낮추면 재현율이 늘어나는 현상
- scikit learn에서 임곗값을 직접 지정할 수는 없지만 예측에 사용한 점수는 확인 가능
  - 분류기의 `decision_function()`을 사용하면 됨
- 적절한 임곗값 정하기
  - 먼저 훈련 세트에 있는 모든 샘플의 점수를 구함
  - 이후 해당 점수로 `precision_recall_curve()` 함수를 사용하여 가능한 모든 임곗값에 대해 정밀도와 재현율을 계산할 수 있음



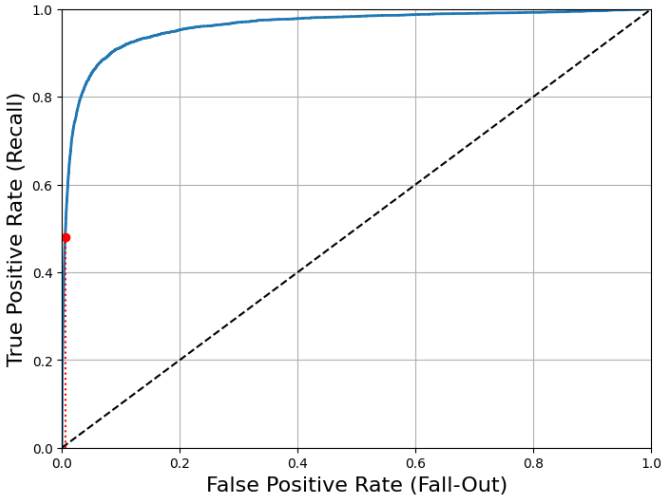
  - 임곗값을 올리더라도 정밀도가 가끔 낮아질 때가 있음(일반적으로는 높아짐)
    - 반면 재현율은 임곗값이 올라감에 따라 줄어들 수밖에 없음
  - 재현율에 대한 정밀도 곡선을 그려서 비교할 수도 있음
    - 정밀도가 급격하게 줄어드는 지점 직전을 정밀도/재현율 트레이드오프로 선택하는 것이 좋음



- `average_precision_score()` 함수를 사용하면 정밀도/재현율 곡선 아래 면적을 계산할 수 있어서 서로 다른 두 모델을 비교할 수 있음
- cf) 재현율이 너무 낮다면 높은 정밀도의 분류기는 전혀 유용하지 않음

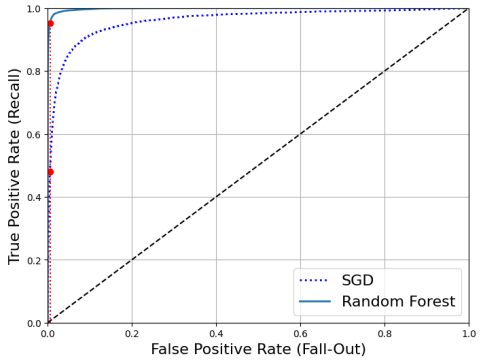
3-3-5. ROC 곡선

- 수신기 조작 특성 곡선(Receiver Operating Characteristic)
- 거짓 양성 비율(FPR = 1 - 특이도)에 대한 진짜 양성 비율(TPR, 재현율)의 곡선
  - 민감도(= 재현율)에 대한 (1 - 특이도) 그래프
  - $FPR = \frac{FP}{FP+TN} = \frac{FP+TN-TN}{FP+TN} = 1 - \frac{TN}{FP+TN} = 1 - TNR$
- `roc_curve()` 함수를 사용해 여러 임곗값에서 TPR과 FPR 계산
  - 레이블과 점수를 입력값으로 다룸
  - 점수 대신 클래스 확률을 전달하는 것도 가능



- ROC AUC 점수
  - `roc_auc_score()` 함수를 통해 구할 수 있음
- ROC 곡선 또한 trade-off가 존재
  - 재현율(TPR)이 높을수록 거짓 양성(FPR)이 늘어남
  - 점선은 완전한 랜덤 분류기(= 무작위 예측)의 ROC 곡선을 의미

    ⇒ 좋은 분류기는 해당 점선에서 최대한 멀리 떨어져 있어야 함(왼쪽 모서리 쪽)



⇒ 랜덤 포레스트 분류기가 SGD 분류기보다 훨씬 좋다고 해석할 수 있음

ㄴ 랜덤 포레스트의 ROC 곡선이 왼쪽 위 모서리에 더 가까움

- 곡선 아래의 면적(area under the curve, AUC)를 측정하여 분류기들을 비교
  - 1에 가까울수록 좋음

cf) 정밀도/재현율 곡선 VS ROC 곡선

- PR 곡선을 사용하는 경우
  - 양성 클래스가 드문 경우
  - 거짓 음성(FN)보다 거짓 양성(FP)이 더 중요할 때
- 그렇지 않으면 ROC 곡선을 사용

3-4. 다중 분류

- 다중 분류기(multi-class classifier)
  - 둘 이상의 클래스를 구분할 수 있는 분류기
  - 일부 알고리즘(SGD, 랜덤 포레스트, 나이브 베이즈 등)은 여러 개의 클래스를 직접 처리할 수 있음

    - 보통의 알고리즘은 이진 분류만 가능 → 이를 여러 개 사용해 다중 클래스를 분류하는 기법 또한 많음
- 다중 분류 방법
    - **OvR 전략**
        - One vs Rest
        - 분류 시 각 분류기의 결정 점수 중에서 가장 높은 것을 클래스로 선택하는 방식
        - ex) 특정 숫자 하나만 구분하는 숫자별 이진 분류기 10개를 훈련시켜 클래스가 10개인 숫자 이미지 분류 시스템을 만듦
        - 대부분의 이진 분류 알고리즘에서 선호하는 방식
    - **OvO 전략**
        - 각 숫자의 조합마다 이진 분류기를 훈련시키는 방식
        - N개의 클래스에 대해 N * (N-1) / 2개의 분류기를 필요로 함
        - 모든 분류기를 통과시켜서 가장 많이 양성으로 분류된 클래스를 선택

          → 각 분류기의 훈련에 전체 훈련 세트 중 구별할 두 클래스에 해당하는 샘플만 필요로 함
    - 사이킷런에서 OvO나 OvR을 사용하도록 강제할 수 있음
        - `OneVsOneClassifier` 또는 `OneVsRestClassifier` 사용

3-5. 에러 분석

- 일반적인 머신러닝 프로젝트는 다음의 순서를 따름
    1. 문제 정의(큰 그림 그리기)
    2. 데이터 수집
    3. 데이터 탐색
    4. 데이터에 내재된 패턴이 머신러닝 알고리즘에 잘 드러나도록 데이터 준비
    5. 여러 모델을 시험해보고 가장 좋은 몇 개의 모델 선택
    6. 모델 튜닝 or 모델 연결(앙상블) → 최선의 솔루션 탐색
    7. 솔루션 출시
    8. 시스템 론칭 & 유지 보수
- 모델의 성능을 향상시키기 위해 **에러 분석**을 활용할 수 있음

    - 만들어진 에러의 종류를 분석
- **방법**
    - 오차 행렬
        - `confusion_matrix`
        - `matplotlib.matshow()` : 오차 행렬을 이미지로 표현 가능

          cf> 사이킷런 0.22 버전부터는 `sklearn.metrics.plot_confusion_matrix()` 함수를 사용할 수 있음
        - 오차 행렬의 각 값을 대응되는 클래스의 이미지 개수로 나누어 에러 비율을 구하여 잘못 분류되는 클래스를 확인할 수 있음

3-6. 다중 레이블 분류(Multi Label Classification)

- 하나의 샘플에 대해 **여러 개**의 클래스를 출력하는 분류기

  ex) 얼굴 인식 분류기 - 같은 사진에 여러 사람이 등장하는 경우
- 평가 방법
    - 여러 지표가 존재

      → F1 score 등

3-7. 다중 출력 분류(Multi Output Classification)

- 다중 레이블 분류에서 한 레이블이 다중 클래스가 될 수 있도록 일반화한 것
    - 두 개 이상의 값을 가질 수 있음
-