Hands-On Machine Learning Seminar

분류

2021.01.14

전병욱
jebuk97@kyonggi.ac.kr

Data Mining Lab.

목차

MNIST

- 고등학생과 미국 인구조사국 직원들이 손으로 쓴 70,000개의 작은 숫자 이미지
- 분류 알고리즘의 성능을 평가할 때 많이 사용.
- 머신러닝 분야의 'Hello World'
- 사이킷런에서 제공 (이외에도 여러 데이터셋 제공)

MNIST 데이터셋에서 추출한 숫자 이미지 ▶

MNIST

- 사이킷런에서 읽어 들인 데이터셋들은 일반적으로 비슷한 딕셔너리 구조를 가짐
  - DESCR 키 : 데이터셋을 설명하는 키
  - data 키 : 샘플이 하나의 행, 특성이 하나의 열로 구성된 배열을 가진 키
  - target 키 : 레이블 배열을 담은 키
- MNIST 데이터셋의 경우 70,000개의 데이터와 784개의 레이블을 가짐.
  - 각 레이블이 각 픽셀의 어두운 정도 (0: 흰색 ~ 255: 검은색)를 나타냄

data 키에 들어있는 이미지 중 첫 번째를 이미지화 ▶

이진 분류기

- 두 개의 클래스를 구분할 수 있는 분류기
- 확률적 경사 하강법 (SGD) :
  - 훈련 샘플을 하나씩 독립적으로 처리
  - 따라서, 매우 큰 데이터셋을 효과적으로 처리 (온라인 학습에 적합)
  - 무작위성을 사용

```
#5인지 아닌지 분류하는 분류기
sgd_clf = SGDClassifier(random_state=42)
sgd_clf.fit(X_train, y_train_5)  #학습 시작

sgd_clf.predict([some_digit])
#맞으면 True, 아니면 False
```

성능 측정

K-겹 교차 검증

- 훈련 세트를 K개의 폴드로 나눔
- StratifiedKFold() : 클래스별 비율이 유지되도록 폴드를 만들기 위해 계층적 샘플링 수행
- 각 폴드에 대해 예측을 만듦
  - 이 때, 나머지 폴드로 훈련시킨 모델을 이용해 폴드에 대한 예측을 만듦
- 적용 예
  - cross_val_score(sgd_clf, X_train, y_train_5, cv=3, scoring="accuracy") :

    3겹의 교차 검증을 사용해 SGDClassifier 모델을 평가

성능 측정

정확도 (Accuracy)

- 정확한 예측의 비율
- 성능 측정 지표로 단독으로 사용되는 것은 선호되지 않음.
  - 불균형한 데이터셋(어떤 클래스가 다른 것 보다 월등히 많은 경우) 특히 선호되지 않음.
  - 불균형한 데이터셋의 예 : 클래스가 5인 숫자 이미지 vs 나머지 숫자 이미지
    - 모두 5가 아니라고 예측한다면, 정확도가 약 90%

성능 측정

오차 행렬

- 분류기 성능을 평가하는 더 좋은 방법
- 클래스 A인 샘플을 클래스 B로 분류한 횟수를 A행 B열에 저장
- 오차 행렬 만들기
  - 실제 타깃과 비교하기 위해 예측값을 만듦 (단, 테스트 세트로 예측을 만들지 않아야 함)

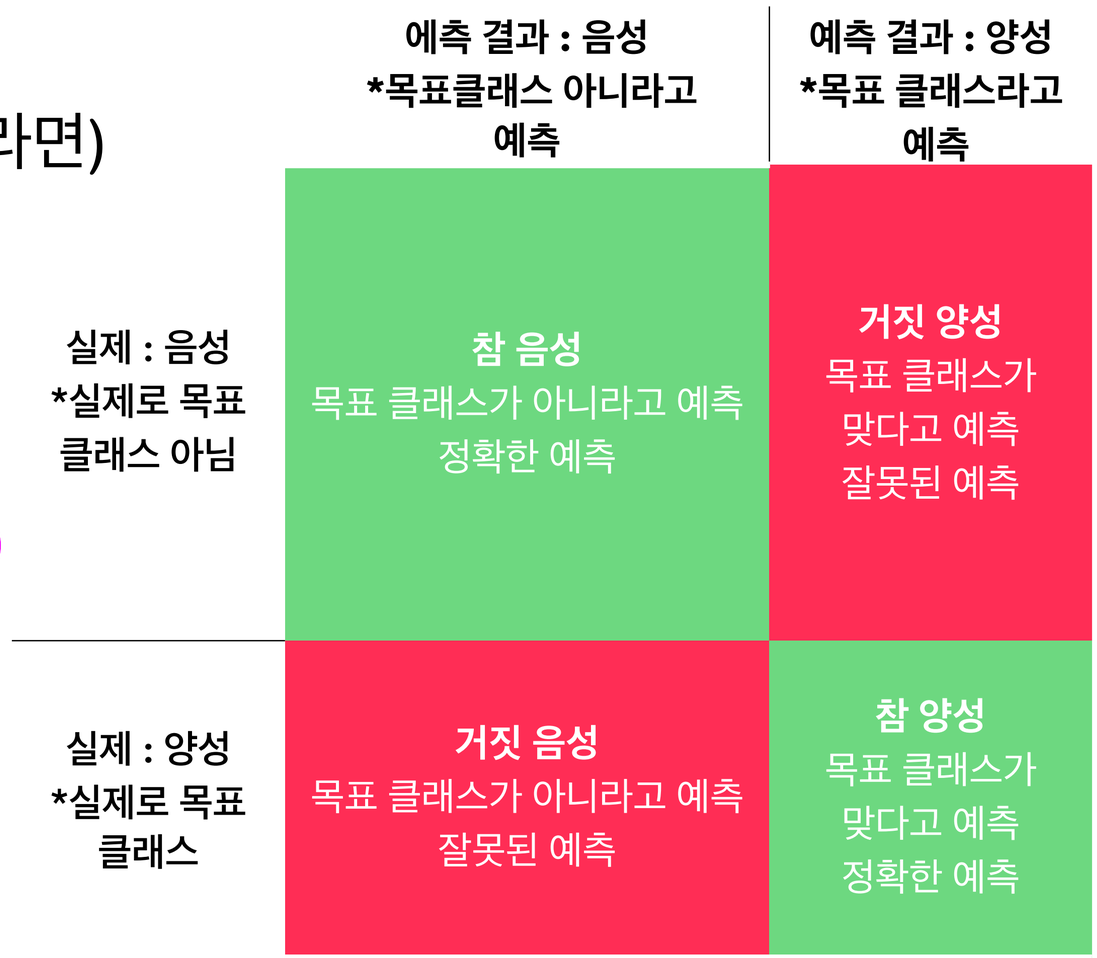
성능 측정

오차 행렬

- 양성 클래스, 음성 클래스
  - 붉은 영역이 없다면 (값이 0이 라면)

    완벽한 분류기

이진 분류기 오차행렬에서 각 클래스의 위치 ▶
(각 영역의 크기는 행렬 값의 크기를 의미합니다)



성능 측정

정밀도

- 분류기의 양성 예측의 정확도
- $\frac{TP}{TP + FP}$ (TP : 참 양성의 수, FP : 거짓 양성의 수)
- 확실한 양성 샘플 하나만 예측하면, 다른 모든 양성 샘플을 무시하고 간단히 완벽한 정밀도를 얻을 수 있음
- 따라서 재현율이라는 지표와 함께 사용하는 것이 일반적

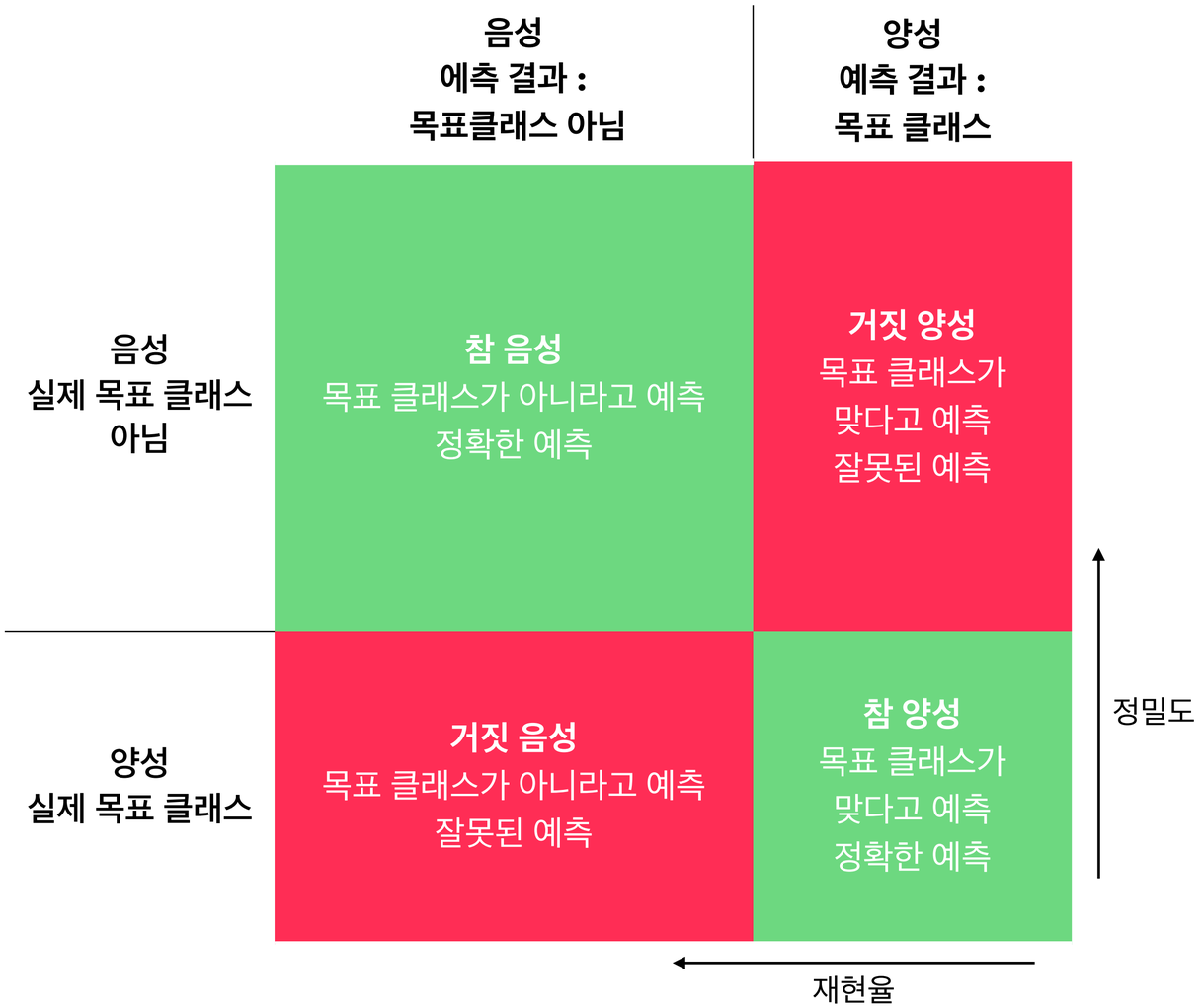
성능 측정

재현율 (민감도, 참양성비율(TPR))

- 분류기가 정확하게 감지한 양성 샘플의 비율
- $\frac{TP}{TP + FN}$ (TP : 참 양성의 수, FN : 거짓 음성의 수, TP + FN : 실제 양성인 샘플 수)

성능 측정

오차행렬 요약


음성
예측 결과 :
목표클래스 아님
양성
예측 결과 :
목표 클래스
음성
실제 목표 클래스
아님
참 음성
목표 클래스가 아니라고 예측
정확한 예측
거짓 양성
목표 클래스가
맞다고 예측
잘못된 예측
양성
실제 목표 클래스
거짓 음성
목표 클래스가 아니라고 예측
잘못된 예측
참 양성
목표 클래스가
맞다고 예측
정확한 예측
정밀도
재현율

성능 측정

정밀도와 재현율

- 정밀도와 재현율을 F_1 점수라는 하나의 숫자로 만들면 편리

$$F_1 = \frac{2}{\frac{1}{\text{정밀도}} + \frac{1}{\text{재현율}}} = 2 * \frac{\text{정밀도} * \text{재현율}}{\text{정밀도} + \text{재현율}} = \frac{TP}{TP + \frac{FN + FP}{2}}$$

- F_1 점수는 정밀도와 재현율의 조화 평균
- 정밀도와 재현율이 비슷할수록 F_1점수가 높음
- f1_score() 함수를 호출하여 사용

성능 측정

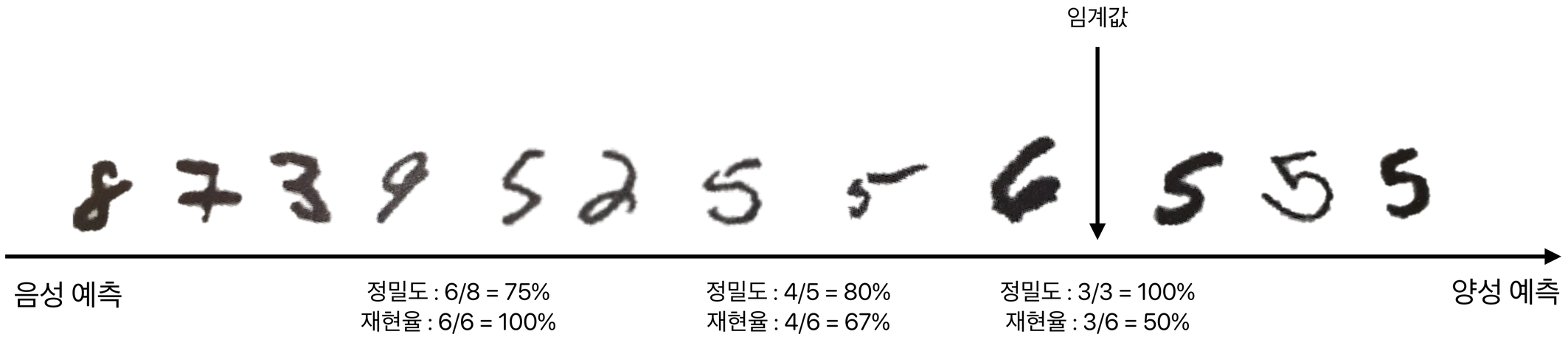
정밀도 / 재현율 트레이드오프

- 정밀도와 재현율을 모두 올리는 것은 불가능
  - 정밀도를 올리면 재현율 감소
  - 재현율을 올리면 정밀도 감소
- 상황에 따라 정밀도가 중요할 수도, 재현율이 중요할 수도 있음.
- 예) 안전한 동영상만 노출하고 싶은 경우
  - 정밀도를 올리는 경우: 안전한 동영상만 노출되지만, 안전한 동영상 중에 누락이 있을 수 있음
  - 재현율을 올리는 경우: 모든 안전한 동영상이 노출되지만, 위험한 동영상이 노출될 수 있음

성능 측정

결정 임계값

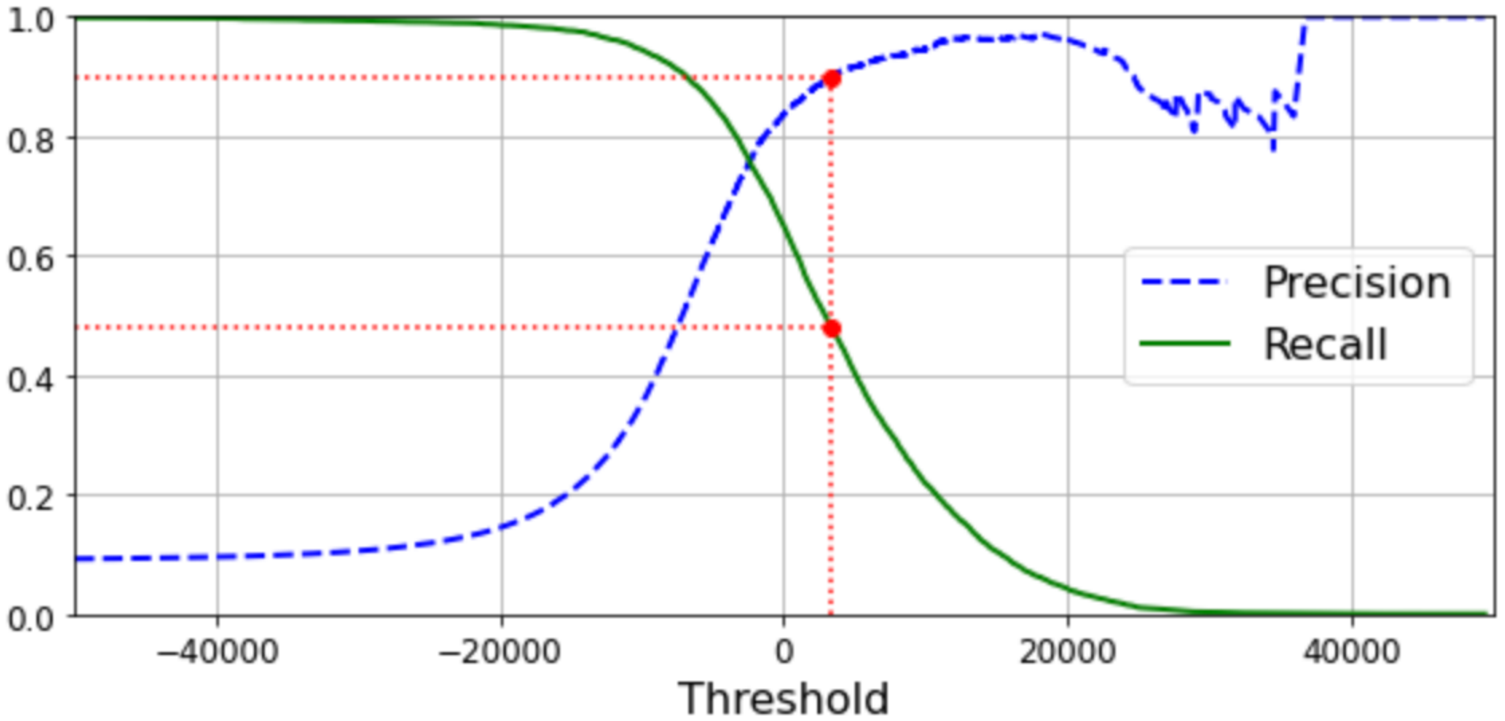
- 분류기는 결정 함수를 사용하여 각 샘플의 점수를 계산
- 결정 함수 : 해당 점수가 임계값보다 크다면 양성 클래스, 그렇지 않다면 음성 클래스에 할당
- Decision_function() 함수를 호출하여 각 샘플의 점수를 얻을 수 있음



▲ 이미지가 5인지 아닌지 분류하는 경우 임계값 변화에 따른 성능 변화

성능 측정

결정 임계값



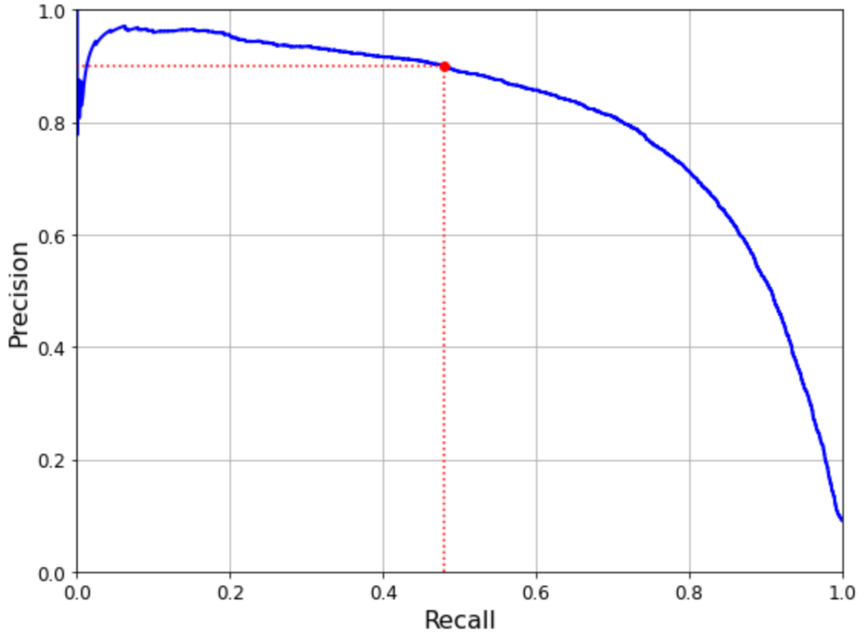
▲ 임계값 변화에 따른 정밀도와 재현율 변화

성능 측정

결정 임계값

- 정밀도만 높히면 재현율이 너무 낮으므로 분류기가 전혀 유용하지 않아짐
- 보통 정밀도가 급격히 줄어들기 시작하는 점을 정밀도/재현율 트레이드오프로 선택
- 정밀도 목표를 설정할 때는 재현율 얼마에서 달성하게 할 것인지도 설정해야함
- 프로젝트에 따라 선택이 달라질 수 있음

재현율에 따른 정밀도 그래프 ▶



성능 측정

ROC(수신기 조작 특성) 곡선

- 이진 분류에서 널리 사용하는 도구
- 거짓 양성 비율 (FPR) 에 대한 참 양성 비율 (TPR)
  - FPR : 양성으로 잘못 분류된 음성 샘플 비율. 1 - TNR (참 음성 비율, 특이도)
  - TPR = 재현율
- roc_curve() 함수를 사용하여 여러 임계값에서 TPR과 FPR 계산

성능 측정

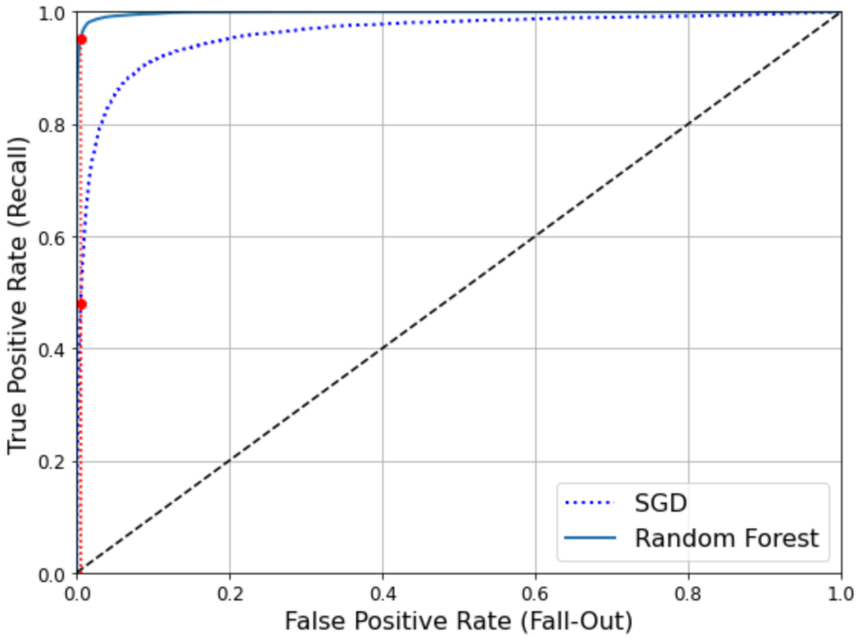
ROC(수신기 조작 특성) 곡선

- 재현율(TPR)이 높을수록 분류기가 만드는 거짓 양성(FPR)이 늘어남
- 그래프에서 검은색 점선은 완전한 랜덤 분류기의 ROC 곡선
- 좋은 분류기 일수록 점선과 최대한 멀리 떨어져서 왼쪽 위로 붙음
- 즉, 곡선 아래의 면적 (AUC)가 높을수록 완벽한 분류기

  (AUC : 0.5 (랜덤 분류기) ~ 1 (완벽한 분류기))
- 아래 그래프에서 더 왼쪽 위 로 붙은 랜덤 포레스트

  분류기가 더 성능이 좋음

SGD 분류기와 랜덤 포레스트 분류기의 ROC 곡선 그래프 ▶



다중 분류

다중 분류기

- 둘 이상의 클래스로 구별할 수 있는 분류기
- SGD, 랜덤 포레스트, 나이브 베이즈 분류기 : 여러 개 클래스 직접 분류 가능
- 로지스틱 회귀, 서포트 벡터 머신 분류기 : 이진 분류만 가능.

  이진 분류기 여러 개로 다중클래스 분류 가능

  - OvR (OvA) 전략 : 각 클래스별로 이진 분류기를 훈련시키고 제일 높은 결정 점수인 클래스를 선택 (클래스 개수 만큼 분류기 필요.)
  - OvO 전략 : 두 개의 클래스로 이루어진 조합마다 이진 분류기 (0과 1 구별, 0과 2 구별 ...)

    를 훈련 시켜서 사용. 가장 많이 양성으로 분류된 클래스를 선택

    (N*(N-1)/2 개의 이진 분류기 필요. N : 클래스 개수)

다중 분류

OvR 전략 vs. OvO 전략

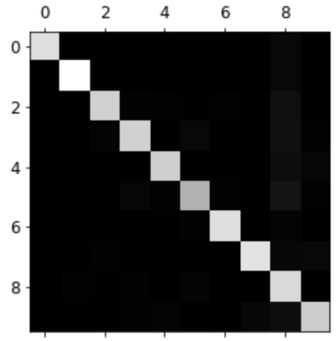
- OvO 전략의 장점
  - 각 분류기 훈련에 전체 훈련 세트 중 구별할 클래스들의 샘플만 필요
  - 훈련 세트 크기에 알고리즘이 민감해서 작은 훈련 세트에서 여러 개의 분류기를 훈련시키는 것이 빠른 경우 선호됨 (예 : 서포트 벡터 머신)
- 그러나 대부분의 알고리즘은 OvR 전략 선호
- 다중 클래스 처리가 가능한 분류기는 전략이 필요 없음
- 사이킷런에서 제공하는 함수로 분류기를 선택하면 자동으로 알맞은 전략 선택
  - OneVsOneClassifier() 나 OneVsRestClassifier() 함수로 전략 강제 선택 가능
- decision_function() 함수로 샘플당 각 클래스에 대한 결정 점수 출력 가능

  (결정 점수가 가장 높은 클래스가 선택됨)

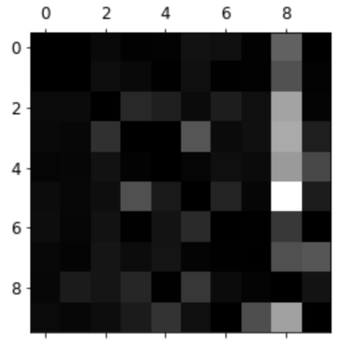
에러 분석

오차 행렬 살펴보기

- cross_val_predict() 함수로 예측을 만들고 confusion_matrix() 함수 호출
- plt.matshow(conf_mx, cmap=plt.cm.gray), plt.show() 함수로 오차행렬 그래프 출력



  - 행은 실제 클래스, 열은 예측한 클래스 색이 밝을수록 해당 예측이 많음
  - 색이 어두울수록 클래스에 대한 이미지 개수가 부족하거나 그 클래스를 잘 분류하지 못한다는 뜻
- 오차 행렬의 각 값을 클래스 별 이미지 개수로 나누어 새로운 에러 비율 그래프를 그린다

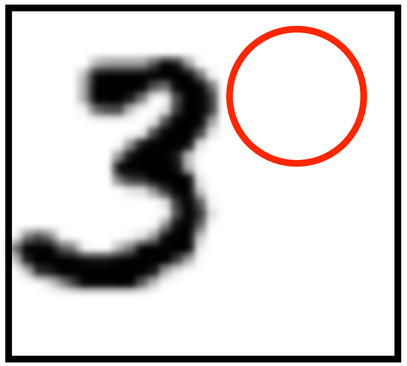


  - (주대각선 (\ ) 은 0으로 채움)
  - 이미지 개수의 영향을 제거하여 순수 에러에 대한 정보를 얻을 수 있음
  - 색이 밝을수록 잘못 분류 되었음을 암시

에러 분석

분류기 성능 향상법

- 오차행렬을 분석하면 분류기 성능 향상에 대한 통찰을 얻을 수 있음
- 분류에 도움이 될 만한 패턴이 존재하는 경우, 그 패턴이 잘 드러나도록 이미지들을 전처리하면 성능이 향상됨
  - MNIST의 경우 3과 5의 주요 차이가 위쪽 선과 아래쪽 호를 이어주는 작은 직선의 위치이므로, 분류기가 이미지의 위치, 회전 방향에 매우 민감하게 3과 5을 분류
  - 그러므로, 숫자를 이미지 중앙에 위치하고 회전되어 있지 않도록 전처리하면 성능 향상



다중 레이블 분류

- 분류기가 하나의 샘플을 여러 클래스로 출력해야 하는 경우
- 특정 분류기만 다중 레이블 분류 지원
- KNeighborsClassifier() 함수로 인스턴스를 만들고 다중 타깃 배열을 만들어서 훈련
- KNeighborsClassifier는 다중 레이블 분류를 지원
- 타깃 레이블 개수만큼 예측 출력도 동시에 출력됨

다중 레이블 분류

다중 레이블 분류기 평가

- 보통 각 레이블의 F_1 점수들의 평균으로 평가
- 프로젝트에 따라 적절한 지표가 다르므로 평가하는 방법은 여러가지
- 어떤 클래스의 비율이 많다면 그 클래스에 대한 분류기의 점수에 더 높은 가중치를 주어 평가
  - 레이블에 클래스의 지지도 (해당 레이블에 속한 샘플 수)를 가중치로 줄 수 있음
  - 단, f1_score() 함수를 사용할 때, average="weighted"로 설정해야 함

다중 출력 분류

(= 다중 출력 다중 클래스 분류)

- 다중 레이블 분류에서 한 레이블이 다중 클래스가 될 수 있도록 일반화 한 것
- 즉, 레이블 하나가 값을 두 개 이상 가질 수 있음

다중 출력 분류

숫자 이미지 잡음 제거

- 이미지의 픽셀 하나가 레이블 하나로 대응됨
- 잡음이 제거된 이미지(픽셀 집합)를 출력하는 분류기의 출력은 각 픽셀의 강도를 담은 배열로 출력
- 픽셀이 레이블 하나이므로 픽셀 배열은 다중 레이블임

▲ 원본 이미지
타깃 레이블

▲ 잡음 추가 이미지
입력 레이블

▲ 잡음 제거 이미지
출력 레이블

감사합니다