BOAZ 멘멘 B조 1주차

CHAPTER 3. 분류

- 함께 첨부한 코드 파일 참조!

18기 분석 강하연

CONTENTS

01. MNIST

02. 이진 분류기 훈련

03. 성능 측정

04. 다중 분류

05. 에러 분석

06. 다중 레이블 분류

07. 다중 출력 분류

MNIST

7만개의 숫자를 손글씨로 나타낸 데이터셋

MNIST

7만개의 숫자를 손글씨로 나타낸 데이터셋

```
y[0]
```

```
'5'
```

뒤 테스트에서 계속 쓰이게 될 데이터로, some_digit에 저장

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

70000개의 데이터를 훈련데이터, 테스트데이터 각각에 60000, 10000 쪼개어 저장

이진 분류기 훈련

- 이미지가 숫자 5인지에 대한 여부 판단

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

* 0 (False) : 숫자 5 아님

* 1 (True) : 숫자 5에 해당

- 확률적 경사 하강법(SGD) 분류기를 이용하여 이미지 분류

```
from sklearn.linear_model import SGDClassifier

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

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

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

> 앞서 봤던 이미지 5를 예측한 결과 True

교차 검증을 사용한 정확도 측정

> SGD 분류기 정확도 : 95% 이상

```
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 ])
```

- cv=3: 3-fold사용
- scoring="accuracy": 정확도를 측정

> **모든** 클래스를 False로 반환하는 모델의 정확도 : 90% 이상

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

```
array([0.91125, 0.90855, 0.90915])
```

클래스가 불균형할 때 정확도 사용의 문제점 ! -> 오차 행렬 확인

오차 행렬

		예측 결과	
		TRUE	FALSE
실제 정답	TRUE	TP (True Positive)	FN (False Negative)
	FALSE	FP (False Positive)	TN (True Negative)

True Positives : 1인 레이블을 1이라 하는 경우 → 관심 범주를 정확하게 분류

False Negatives : 1인 레이블을 0이라 하는 경우 → 관심 범주가 아닌것으로 잘못 분류

False Positives : 0인 레이블을 1이라 하는 경우 → 관심 범주라고 잘못 분류

True Negatives : 0인 레이블을 0이라 하는 경우 → 관심 범주가 아닌것을 정확하게 분류

오차 행렬 교차 검증을 이용하여 정확도 대신 오차 행렬을 확인

> SGD classifier의 교차검증 폴드(cv=3)에 대한 오차행렬

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

```
array([[53892,   687],
       [ 1891,  3530]])
```

True Positives : '5'로 정확히 분류

False Negatives : '5 아님'으로 잘못 분류

False Positives : '5'로 잘못 분류

True Negatives : '5 아님'으로 정확히 분류

정밀도 / 재현율

$$Precision = \frac{TP}{TP + FP}$$

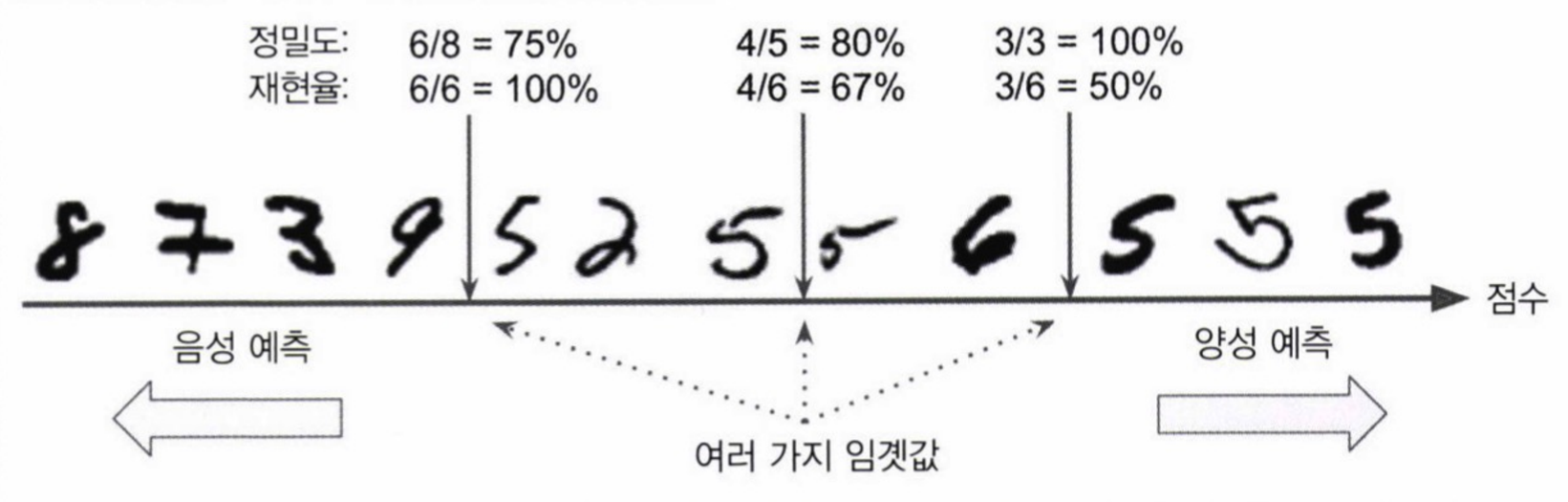
True로 예측한 것 중에 True의 비율

$$Recall = \frac{TP}{TP + FN}$$

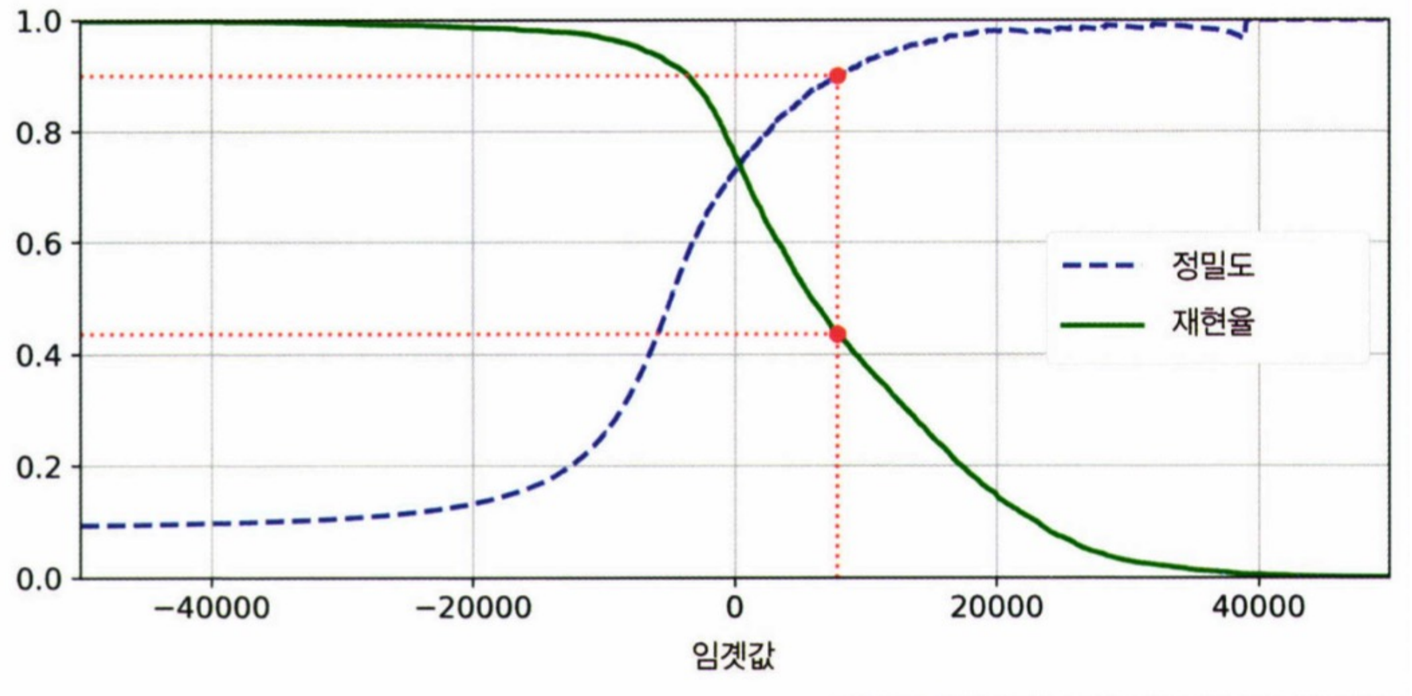
True인 것 중에 True로 예측한 비율

정밀도 / 재현율 Trade-off

- 각 샘플에 대해 분류기는 결과값을 score로 나타냄
- 각 score에 대해 임계값(threshold, default=0.5)을 정해서 클래스 분류
- 아래 그림은 임계치를 다르게 주면서 정밀도와 재현율의 변화를 나타냄(trade-off가 존재)



정밀도 / 재현율 Trade-off



임계값이 달라지며 정밀도와 재현율 사이 Trade off가 일어남을 확인할 수 있음

F1 Score 정밀도와 재현율을 함께 고려

F1 score은 정밀도와 재현율의 조화평균으로 데이터셋의 라벨이 불균형일 때 유용

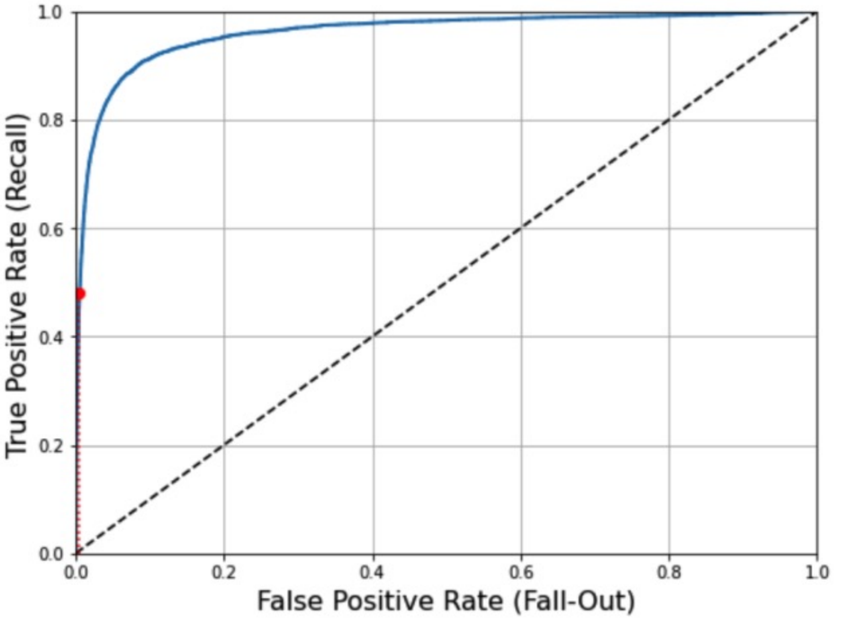
$$\text{F1 Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

정밀도와 재현율을 다르게 반영하고 싶으면 가중치를 적용하는 식으로 사용하기도 함

ROC 곡선 거짓 양성 비율 (FPR)에 대한 진짜 양성 비율(TPR, 재현율)의 곡선

$$TPR = \frac{TP}{FN+TP}$$

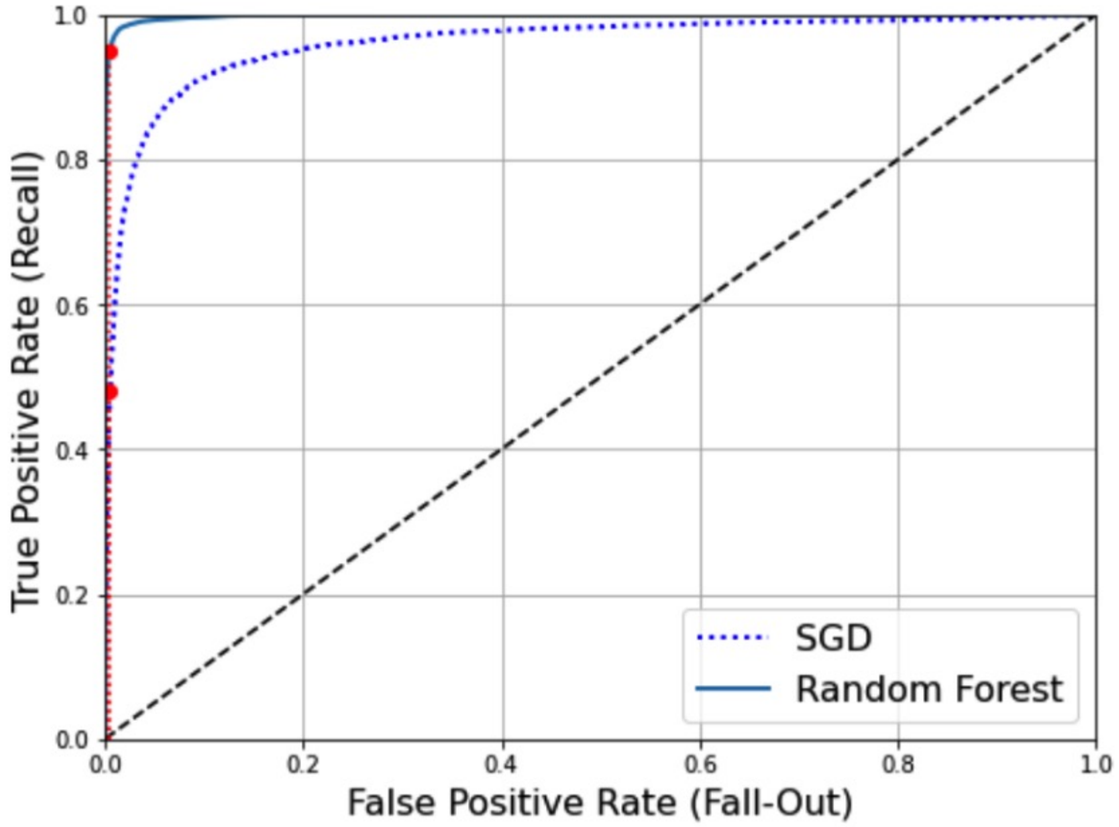
$$FPR = \frac{FP}{TN+FP}$$



- 다양한 threshold에 대한 이진분류기의 성능을 한번에 표시한 곡선
- 재현율(TPR)이 높을수록 거짓 양성(FPR)이 늘어남
- 랜덤분류기의 ROC(중간 점선)에서 최대한 멀리 떨어져야 좋은 분류기

AUC ROC 곡선 아래의 면적, 분류기 비교에 사용

< SGD, 랜덤 포레스트 분류기의 ROC 곡선 >



→ 한눈에 보기에도 랜덤 포레스트 분류기의 곡선이 더 이상적임

→ AUC 값도 랜덤포레스트가 더 높음

```
SGD의 AUC: 0.9604938554008616
Random Forest의 AUC: 0.9983436731328145
```

다중 분류 둘 이상의 클래스를 분류

- 이진 분류를 기본으로 하는 모델은 다음과 같은 전략으로 다중 분류를 수행
- OvR(One-versus-the-Rest)
  - 클래스마다 이진 분류기를 만들어 가장 높은 결정점수를 낸 클래스 선택
  - 대부분의 이진 알고리즘에서 사용

  * ex) 0과 0 아닌 것, 1과 1 아닌 것 ..

- OvO(One-versus-One)
  - 가능한 두 개의 모든 클래스 조합에 대해 이진 분류기를 만듦
  - 가장 양성으로 많이 분류된 클래스를 선택

  * ex) 0과 1, 0과 2, 1과 2 ..

SVC - OvO

- OvO 훈련 후 5를 넣어 예측해 본 결과 알맞게 분류

```
svm_clf = SVC()
svm_clf.fit(X_train[:1000], y_train)
svm_clf.predict([some_digit])
```

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

- 각 클래스 별 점수 확인

```
array([[ 1.75828215,  2.74956232,  6.13809559,  8.2853702 , -0.28728967,
         9.30119996,  0.74228825,  3.79256174,  7.20847395,  4.85762716]])
```

SVC - OvR

강제로 SVM에 OvR 사용하려면 OneVsRestClassifier 사용

```
from sklearn.multiclass import OneVsRestClassifier
ovr_clf = OneVsRestClassifier(SVC(gamma="auto", random_state=42))
ovr_clf.fit(X_train[:1000], y_train[:1000])
```

```
OneVsRestClassifier(estimator=SVC(gamma='auto', random_state=42))
```

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

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

숫자 5를 넣어 예측한 결과 바르게 분류

```
len(ovr_clf.estimators_)
```

```
10
```

OvR이므로, 모델의 수 10개

SGD, Random Forest

강제로 SGD classifier에 OvR 사용하려면?

→ SGD와 Random Forest는 다중 분류를 지원하므로 OvO, OvR 적용할 필요 없음

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

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

간단한 스케일링(StandardScaler)을 이용하여 성능 향상

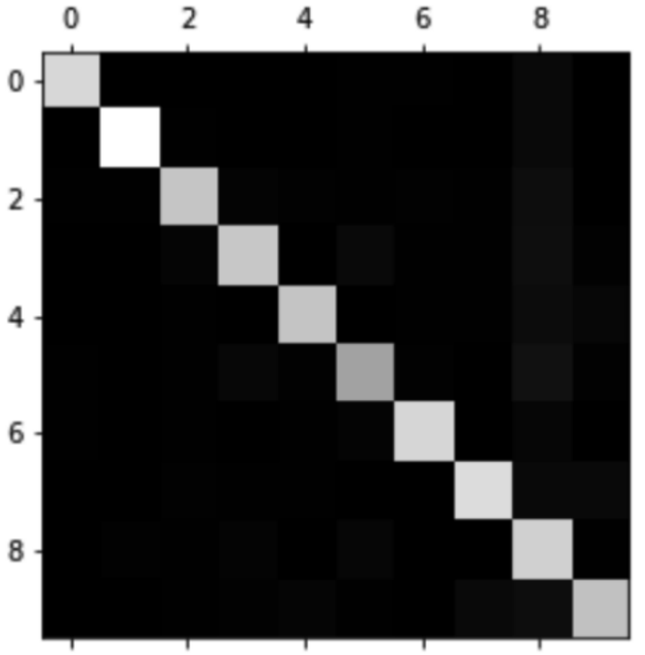
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()

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])
```

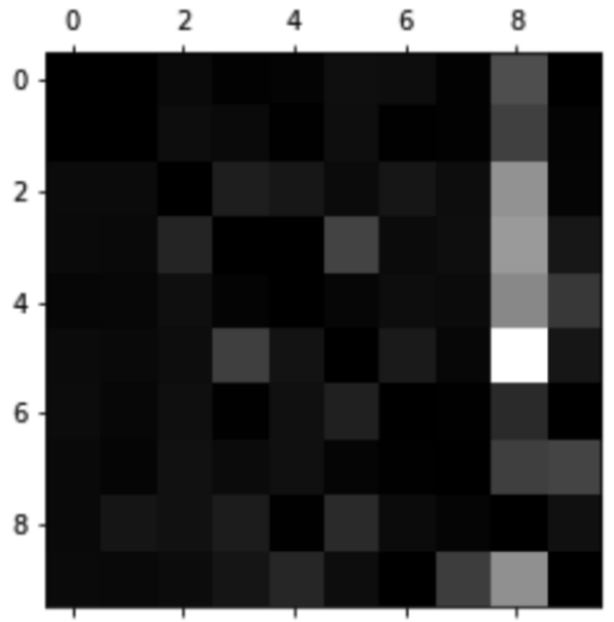
에러 분석

SGD의 cross validation 결과로 오차 행렬 확인



숫자 5가 다른 숫자보다 조금 더 어두워 보임→ 숫자 5를 다른 숫자만큼 잘 분류하지 못함

에러 분석



> 에러에 집중하기 위해 대각선을 채운 그래프

- 8행은 실제 8이 적절히 8로 분류되었음을 보여줌
- 8열은 8이 아닌 이미지가 8로 잘못 분류된 것이 많음을 보여줌

→ 오차 행렬 분석을 통해 분류기 성능 향상의 방안을 모색할 수 있음

다중 레이블 분류 각 샘플에 여러가지 클래스가 존재하는 경우

- 다중 레이블 분류를 지원하는 KNeighborsClassifier 모델을 사용

```
y_train_large = (y_train >= 7) # 7보다 같거나 큰지
y_train_odd = (y_train % 2 == 1) # 홀수인지
y_multilabel = np.c_[y_train_large, y_train_odd]

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

knn_clf.predict([some_digit])
```

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

- 샘플이 홀수인지의 여부
- 7보다 같거나 큰지의 여부
- np.c_: 두 array를 합쳐줌

5를 넣어 예측한 결과가 (False, True)이므로 올바르게 분류됨

다중 레이블 분류의 평가

- 다중 레이블 분류기를 평가하는 방법은 모델과 목적에 따라 다름
  - ex) 각 레이블의 F1 score를 구하고 레이블에 대한 가중치를 적용한 평균 점수 계산

    가중치 예시: 샘플 수가 많은 클래스에 더 큰 가중치 부여
- 아래 예시는 모든 레이블에 가중치를 동일하게 줌(average="macro")

```
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")
```

```
0.976410265560605
```

다중 출력 분류

- 다중 레이블 분류에서 한 레이블이 다중 클래스가 될 수 있음
- MNIST의 여러개의 픽셀, 픽셀마다 강도를 나타내는 숫자 데이터는 다중 출력 분류에 해당

```
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]])
```

감사합니다