Classification

Samkeun Kim <skim@hknu.ac.kr>

http://cyber.hankyong.ac.kr

가장 흔한 Supervised 학습 태스크:

- 회귀(값 예측)
- 분류(클래스 예측)

이제 분류 시스템에 대해 알아 보자.

MNIST

MNIST data:

- 미국 고등학생과 인구조사국 직원들이 필기한 7만 개의 작은 숫자 이미지 세트인 MNIST 데이터 세트
- 각 이미지에는 해당 숫자가 레이블로 붙어 있다.
- 이 세트는 머신 러닝의 "hello world"라고 불릴 정도로 많이 연구되었다.
- 사람들이 새로운 분류 알고리즘을 제안할 때마다 MNIST에서는 어떻게 수행되는지 궁금해하며 머신 러닝을 배우는 사람은 누구나 이 문제를 다루어 본다.

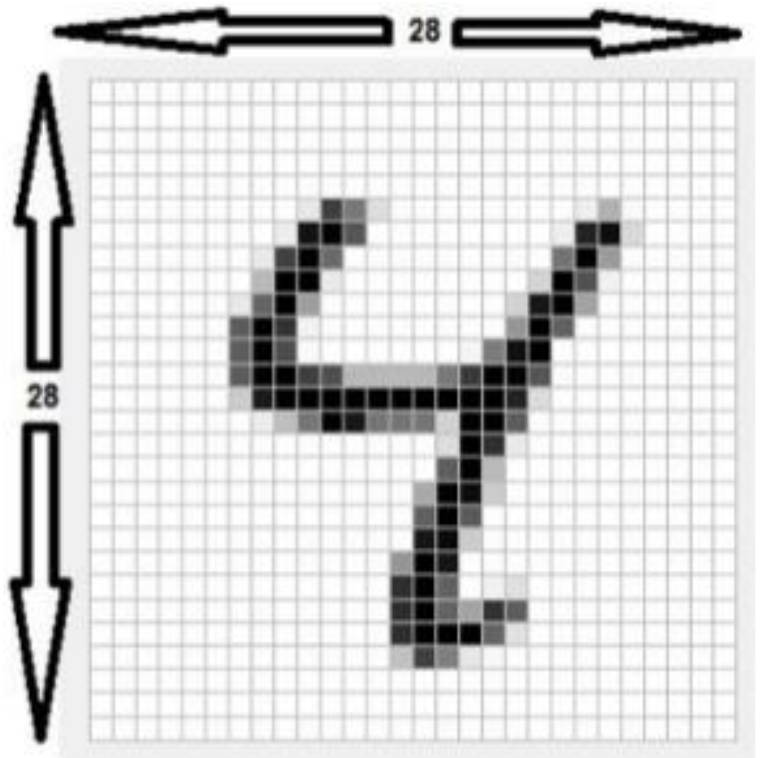
Scikit-Learn은 널리 사용되는 데이터 세트를 다운로드할 수 있는 많은 도우미 기능을 제공한다.

- MNIST는 그중 하나이다.

다음 코드는 MNIST 데이터 세트를 가져온다:

MNIST(Mixed National Institute of Standards and Technology database)

우체국에서 수기로 작성한 우편번호(숫자: 0-9)를 인식하기 위한 dataset



아래 코드는 MNIST dataset을 fetch한다: (MNIST data set을 가져올 수 있도록 라이브러리화 되어 있음)

```
>>> from sklearn.datasets import fetch_openml
>>> mnist = fetch_openml('mnist_784', version=1)
>>> mnist.keys()
dict_keys(['data', 'target', 'feature_names', 'DESCR', 'details',
           'categories', 'url'])
```

가져온 배열을 살펴 보자:

```
>>> X, y = mnist["data"], mnist["target"]
>>> X.shape
(70000, 784)
>>> y.shape
(70000,)
```

- 70,000개의 이미지가 있다.
- 각 이미지는 784개의 특성을 갖는다(28 x 28 pixels).
- 각 픽셀 값의 범위는 0 ~ 255 사이의 값이다.

데이터 셋에서 1개의 숫자를 가져와서 28 x 28 배열로 reshape한 다음, Matplotlib의 imshow() 함수를 이용하여 표시해 보자:

```
import matplotlib as mpl
import matplotlib.pyplot as plt

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

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

레이블이 스트링이다:

```
>>> y[0]
'5'
```

대부분 ML 알고리즘은 숫자를 기대한다:

```
>>> y = y.astype(np.uint8)
```

MNIST dataset에서 좀 더 많은 이미지를 가져와서 표시해보자:

A few digits from the MNIST dataset

잠깐! 항상 test set을 별도로 보관해 두어야 한다.

- MNIST 데이터 셋 => training set과 test set을 미리 분리해 둠
- 끝부분 10,000개의 이미지가 test set

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

- 중요: training set을 섞어야 한다!! => `permutation()`

```
import numpy as np

shuffle_index = np.random.permutation(60000)
X_train, y_train = X_train[shuffle_index], y_train[shuffle_index]
```

Training a Binary Classifier

숫자 5를 구별하는 "5-detector"를 만들어보자: binary classifier

```
y_train_5 = (y_train == 5)  # True for all 5s, False for all other digits.
y_test_5 = (y_test == 5)
```

Scikit-Learn's Stochastic Gradient Descent (SGD) 클래스 사용:

```
from sklearn.linear_model import SGDClassifier

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

SGD를 이용하여 숫자 5의 이미지 탐색:

```
>>> sgd_clf.predict([some_digit])
array([ True], dtype=bool)
```

← 이미지가 숫자 5임을 의미

Performance Measure

Classifier를 평가하는 일은 Regressor를 평가하는 것보다 심각하게 트릭이 필요하다.

- 많은 새로운 개념 및 용어를 습득해야

Measuring Accuracy Using Cross-Validation

모델을 평가하기 위한 좋은 방법 => Cross-Validation 이용

- `cross_val_score()` 함수를 이용하여 `SGDClassifier` 모델 평가
- 3개의 fold를 갖는 K-fold cross-validation 이용하여 모델 평가:

```
>>> from sklearn.model_selection import cross_val_score
>>> cross_val_score(sgd_clf, X_train, y_train_5, cv=3, scoring="accuracy")
array([0.96355, 0.93795, 0.95615])
```

Wow! Above 93% accuracy on all cross-validation folds.

이제 더 나가기 전에 모든 이미지를 "not-5" 클래스라고 분류하는 멍청한 classifier를 만들어보자:

```
from sklearn.base import BaseEstimator

class Never5Classifier(BaseEstimator):
    def fit(self, X, y=None):
        pass
    def predict(self, X):
        return np.zeros((len(X), 1), dtype=bool)
```

모든 이미지를 '5'가 아니라고 분류하는 Classifier

이 모델의 accuracy를 알아 맞출 수 있는가?

```
>>> 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])
```

- 90% 이상이다. 이미지의 약 10%만이 5이기 때문에 모든 경우에 5가 아니라고 해도 약 90%는 맞출 수 있다!

⇒ 이것이 Accuracy가 classifier의 performance measure로 환영 받지 못하는 이유~!!

⇒ 특히 **skewed dataset**을 다룰 때(어떤 클래스가 다른 클래스보다 훨씬 더 많을 때) 더 그렇다!

Confusion Matrix

Classifier의 performance를 평가하기 위한 훨씬 더 좋은 방법: **confusion matrix**

- 클래스 A의 인스턴스를 클래스 B로 분류한 횟수 카운트!

예) Classifier가 5의 이미지를 3으로 잘못 분류한 횟수를 알기 위해서는 confusion matrix의 5행 3열을 보면 된다(행: 실제 이미지, 열: 예측 이미지).

Confusion matrix 계산을 위해:

- 예측 결과와 실제 타겟 필요
- Test set에서 예측해야 하지만 지금은 손대지 말자.
- 대신 `cross_val_predict()` 함수를 사용하자.

```
from sklearn.model_selection import cross_val_predict

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

`cross_val_predict()` 함수:

- `cross_val_score()` 함수와 마찬가지로 K-fold cross-validation 이용
- Score를 반환하는 대신 각 테스트 fold에서 도출한 prediction을 반환

Confusion matrix 계산:

```
>>> from sklearn.metrics import confusion_matrix
>>> confusion_matrix(y_train_5, y_train_pred)
array([[53057,  1522],
       [ 1325,  4096]])
```

Negative class

Positive class

- 각 행: actual class, 각 열: predicted class
- 첫 번째 행: non-5 이미지(negative class) 검토
  - ✓ 53,057 이미지 => non-5로 정확하게 분류 (true negatives, TN)
  - ✓ 1,522 이미지 => 5로 잘못 분류 (false positives, FP)
- 두 번째 행: 5 이미지(positive class) 검토
  - ✓ 1,325 이미지 => non-5로 잘못 분류 (false negatives, FN)
  - ✓ 4,096 이미지 => 5로 정확하게 분류 (true positives, TP)

완벽한 classifier의 confusion matrix => true positives/true negatives만 가짐:

```
>>> confusion_matrix(y_train_5, y_train_perfect_predictions)
array([[54579,     0],
       [    0,  5421]])
```

더 간결한 척도를 선호한다 => **Precision**

- Positive 예측의 정확성(accuracy):

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

TP: true positives의 개수, FP: false positives의 개수

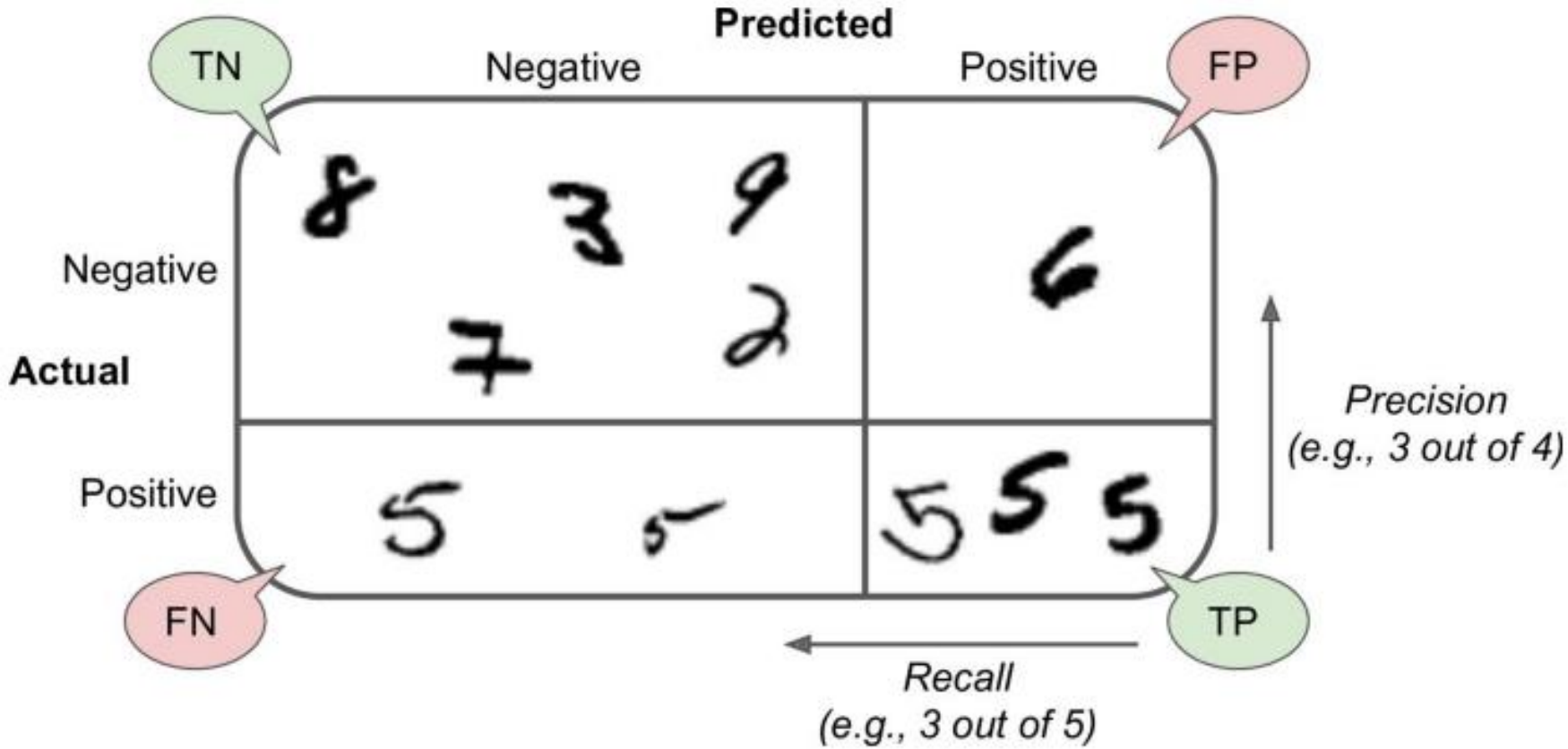
Perfect precision을 갖게 하는 단순한 방법:

- 단 1개의 예측만 수행 (Precision = 1/1 = 100%)
- 이 문제를 해결하는 또 다른 Metric 필요

Recall => Classifier에 의해 정확하게 찾아진 Positive 인스턴스의 비율:

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

FN: false negatives의 개수



An illustrated confusion matrix

Precision and Recall

```
array([[53057,  1522],
       [ 1325,  4096]])
```

Scikit-Learn => precision_score, recall_score 함수 제공:

```
>>> from sklearn.metrics import precision_score, recall_score
>>> precision_score(y_train_5, y_train_pred) # == 4096 / (4096 + 1522)
0.7290850836596654
>>> recall_score(y_train_5, y_train_pred) # == 4096 / (4096 + 1325)
0.7555801512636044
```

이제 5-detector를 보면 그렇게 똑똑해 보이진 않는다:

- Precision => 약 72.9%, Recall => 75.6% (전체 5 중에서)

종종 precision과 recall을 하나로 합친 metric이 편리: F1 score

- 특히 2개의 classifier를 비교해야 할 때

Precision and Recall

$$\text{precision} = \frac{TP}{TP + FP} \qquad \text{recall} = \frac{TP}{TP + FN}$$

F1 score (harmonic mean):

- 평균은 모든 값을 동일하게 취급하지만, 조화 평균은 낮은 값에 훨씬 더 많은 가중치를 준다.
- 따라서 precision과 recall 값이 high일 때만 F1 score 값이 high!

$$F_1 = \frac{2}{\frac{1}{\text{precision}} + \frac{1}{\text{recall}}} = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}} = \frac{TP}{TP + \frac{FN + FP}{2}}$$

```
>>> from sklearn.metrics import f1_score
>>> f1_score(y_train_5, y_train_pred)
0.7420962043663375
```

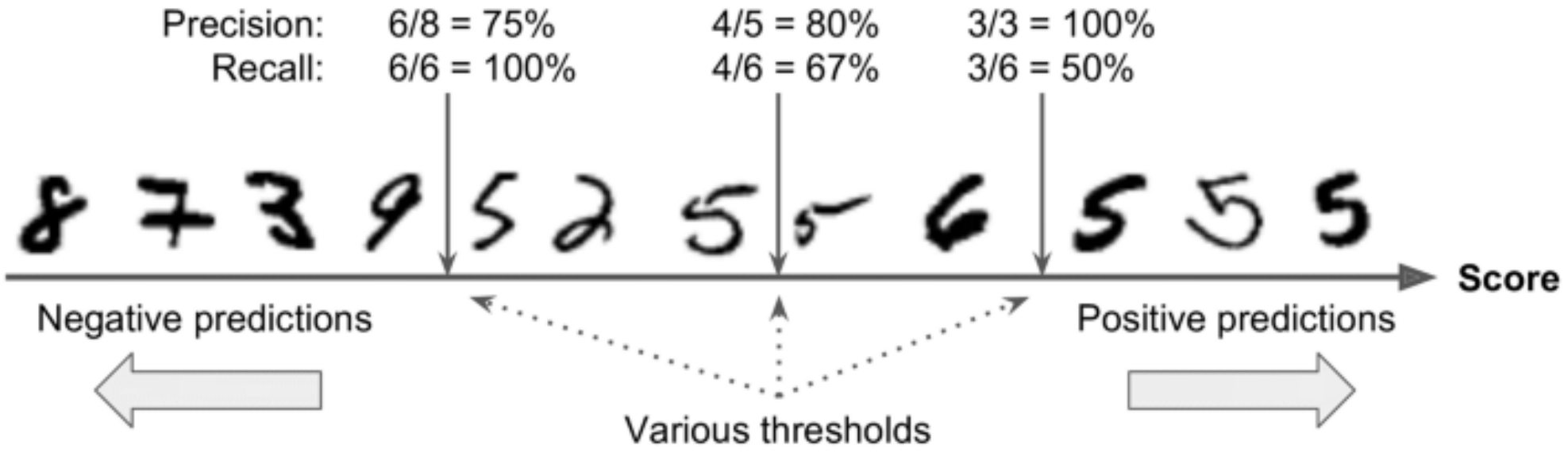
- F1 score => 보통 precision과 recall 값이 비슷할 때 유효, 그러나 상황에 따라 다를 수 있다.
  - A. 어린이에게 안전한 비디오를 탐색하는 classifier일 경우 => low recall, high precision 선호
  - B. 물건 도둑을 찾아내는 classifier일 경우 => high recall, low precision 선호

※ 불행하게도 위 A, B 방식을 함께 가질 수는 없다: precision을 높이면 recall이 줄어든다. 그 역도 마찬가지!

Precision/Recall Tradeoff

Tradeoff 이해를 위해 `SGDClassifier`가 어떻게 분류 결정을 내리는지 살펴보자:

1. 각 인스턴스에 대해 decision function에 기반한 score 계산
2. Score가 임계 값(threshold) 보다 크면 해당 인스턴스를 positive 클래스로 분류
3. 그렇지 않으면 negative 클래스로 분류



Scikit-Learn => 임계 값을 직접 설정 못함 => 대신 예측하는데 사용한 score에 접근 가능

- Classifier의 `predict()` 메소드 호출 대신 `decision_function()` 메소드 호출
- `decision_function()` 메소드 => 각 인스턴스에 대한 score 반환
- 임의의 임계 값을 이용하여 score에 기반한 예측 수행

```
>>> y_scores = sgd_clf.decision_function([some_digit])
>>> y_scores
array([ 161855.74572176])
>>> threshold = 0
>>> y_some_digit_pred = (y_scores > threshold)
array([ True], dtype=bool)
```

`SGDClassifier` => 임계 값 0을 사용 => `predict()` 메소드와 동일한 결과(즉 `True`) 반환

- 임계 값을 높여보자:

```
>>> threshold = 8000
>>> y_some_digit_pred = (y_scores > threshold)
>>> y_some_digit_pred
array([False])
```

- 임계 값을 높이면 recall이 작아진다.
- 임계 값이 0일 때는 찾았지만, 임계 값이 8,000일 때는 찾지 못한다 (즉 False)

어떤 임계 값을 사용할 것인가? => 먼저 training set의 모든 인스턴스의 score를 얻어와야 한다:

- `cross_val_predict()` 함수 이용 => prediction 대신 decision score를 반환하게 한다!

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

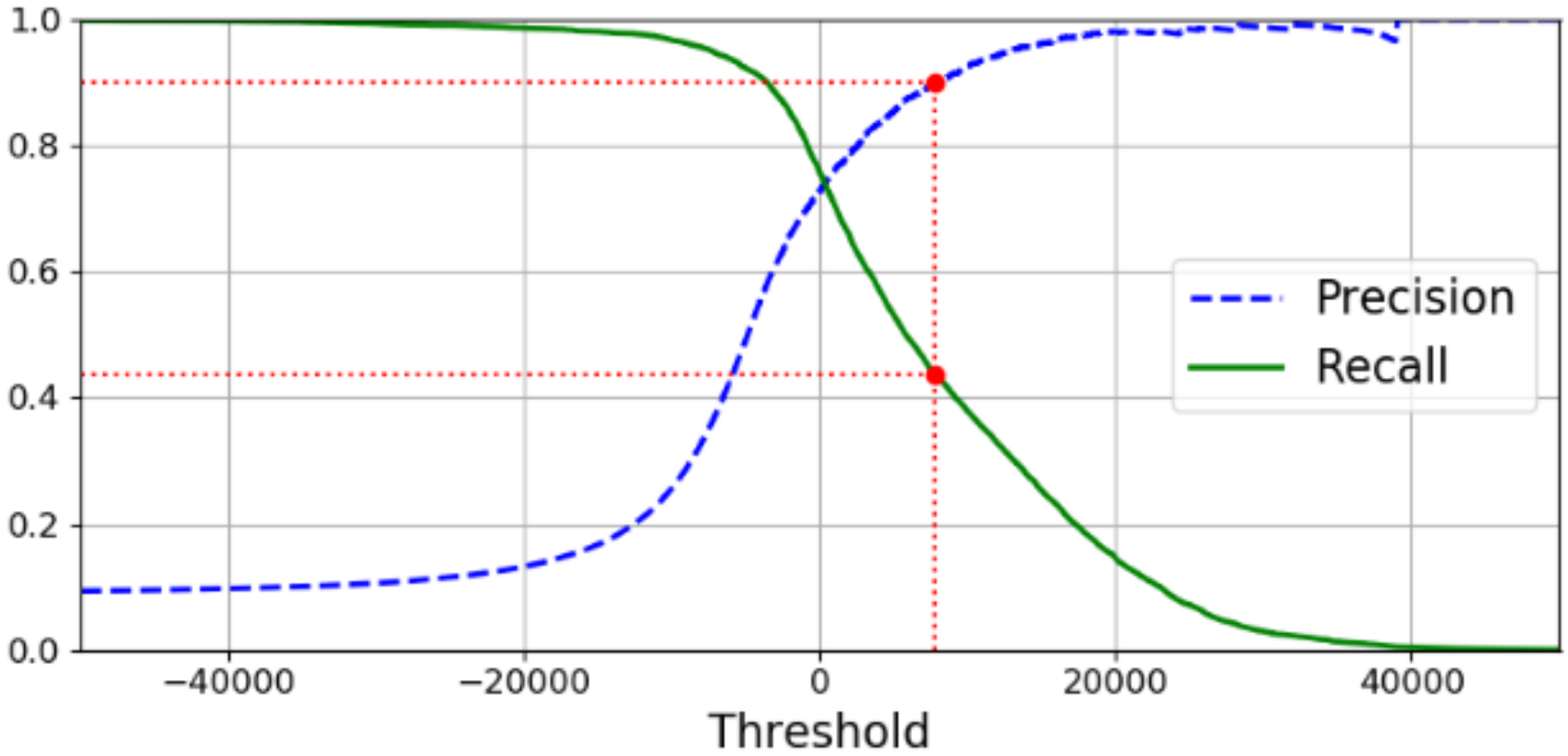
precision_recall_curve() 함수 이용 => 모든 가능한 임계 값에 대해 precision과 recall 계산 가능:

```
from sklearn.metrics import precision_recall_curve

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

Matplotlib를 이용하여 precision과 recall을 임계 값의 함수들로 plot 가능!

```
def plot_precision_recall_vs_threshold(precisions, recalls, thresholds):
    plt.plot(thresholds, precisions[:-1], "b--", label="Precision")
    plt.plot(thresholds, recalls[:-1], "g-", label="Recall")
    [...] # highlight the threshold and add the legend, axis label, and grid

plot_precision_recall_vs_threshold(precisions, recalls, thresholds)
plt.show()
```

Precision/Recall Tradeoff

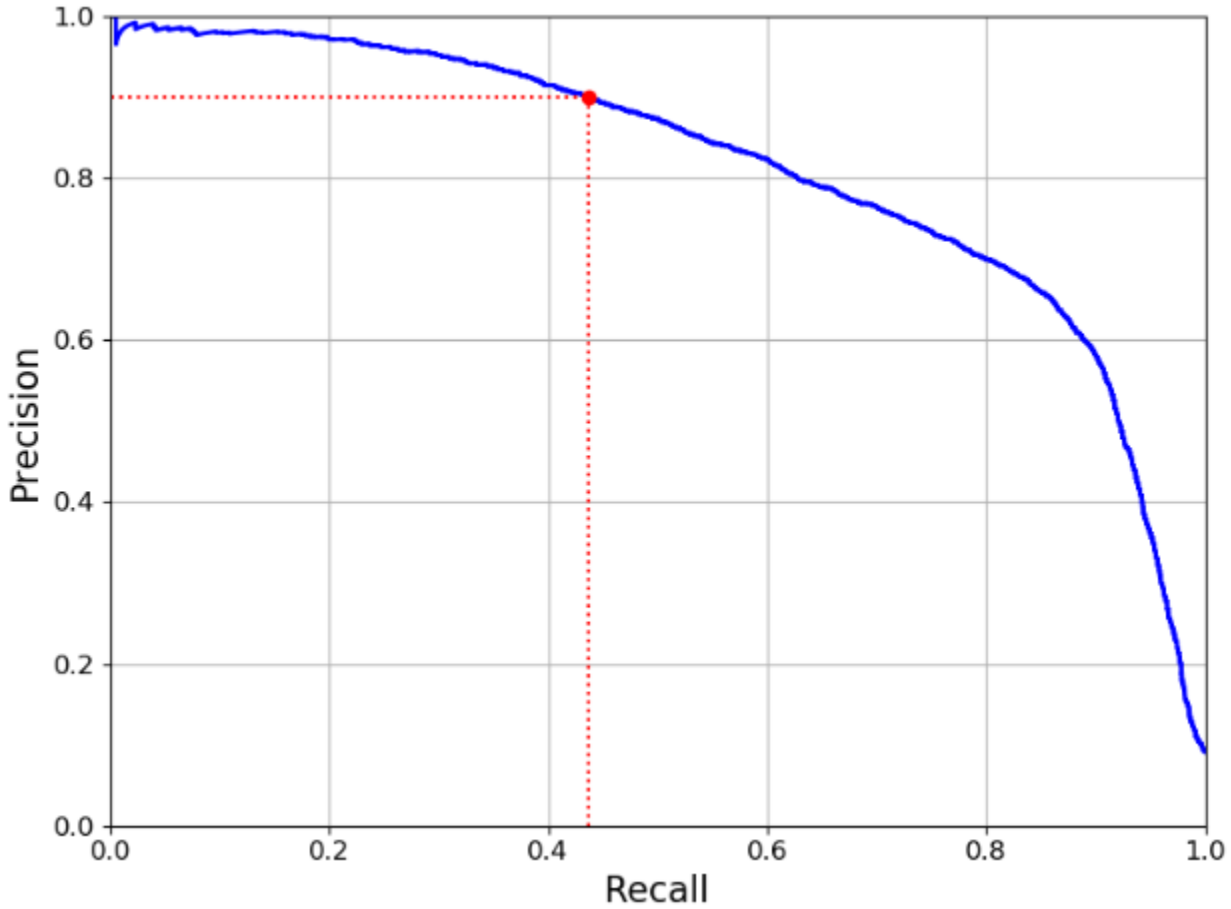


Precision and recall versus the decision threshold

Best precision/recall tradeoff을 주는 임계 값을 선택 가능!

좋은 precision/recall tradeoff을 선택하기 위한 또 다른 방법:

- Precision vs. recall을 직접 plotting



90% precision classifier를 만들어 보자:

```
threshold_90_precision = thresholds[np.argmax(precisions >= 0.90)] # ~7816


y_train_pred_90 = (y_scores >= threshold_90_precision)
```

예측에 대한 precision과 recall 체크:

```
>>> precision_score(y_train_5, y_train_pred_90)
0.9000380083618396
>>> recall_score(y_train_5, y_train_pred_90)
0.4368197749492714
```

Great! You have a 90% precision classifier.

> **TIP**
>
> If someone says "let's reach 99% precision," you should ask, "at what recall?"

Multiclass(multinomial) Classification

Binary classifier => 2개의 클래스 분류 가능

Multiclass classifier => 2개 이상의 클래스 분류 가능

Some 알고리즘:

- Random Forest classifier => 직접 다중 클래스 분류 가능
- Support Vector Machine classifier => 순수한 binary classifier
- 여러 개의 binary classifier를 이용하여 multiclass 분류 작업을 수행하게 할 수 있는 다양한 전략

Digit 이미지를 10개의 클래스(0~9)로 분류할 수 있는 시스템 생성 방법:

- 10개의 binary classifier를 학습 시키는 것
- 각 digit에 대해 1개의 classifier (0-detector, 1-detector, 2-detector 등)
- 해당 이미지에 대한 각 classifier의 decision score를 얻어서 가장 높은 score를 출력하는 클래스 선택
- One-versus-all (OvA) 또는 One-versus-the-rest (**OvR**) 전략이라 한다.

Scikit-Learn => Multiclass 분류 작업을 위해 Binary 분류 알고리즘을 사용하려고 할 때를 탐지하여, 자동으로 OvA 실행!

- `sklearn.svm.SVC` 클래스를 사용하여 SVM Classifier에 적용해 보자:

```
>>> from sklearn.svm import SVC
>>> svm_clf = SVC()
>>> svm_clf.fit(X_train, y_train) # y_train, not y_train_5
>>> svm_clf.predict([some_digit])
array([5], dtype=uint8)
```

- Scikit-Learn => 실제로 10개의 binary classifier를 학습시킴 => 가장 높은 score를 가진 클래스 선택:
- `decision_function()` 메소드를 호출한다면:

```
>>> some_digit_scores = svm_clf.decision_function([some_digit])
>>> some_digit_scores
array([[ 2.92492871,  7.02307409,  3.93648529,  0.90117363,  5.96945908,
         9.5       ,  1.90718593,  8.02755089, -0.13202708,  4.94216947]])


>>> np.argmax(some_digit_scores)
5
```

`RandomForestClassifier` 학습시키는 일 => 매우 쉽다!

```
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")
```

```
>>> forest_clf.fit(X_train, y_train)
>>> forest_clf.predict([some_digit])
array([ 5.])
```

- Random Forest classifier => 인스턴스들을 multiple 클래스로 직접 분류할 수 있다.

```
>>> forest_clf.predict_proba([some_digit])
array([[ 0.1,  0. ,  0. ,  0.1,  0. ,  0.8,  0. ,  0. ,  0. ,  0. ]])
```

Classifier를 평가해 보자: `cross_val_score()` 함수를 이용하여 `SGDClassifier`의 정확도를 평가

```
>>> cross_val_score(sgd_clf, X_train, y_train, cv=3, scoring="accuracy")
array([0.8489802 , 0.87129356, 0.86988048])
```

- 모든 test fold에 대해 84% 이상의 정확도를 얻었다.
- Scaling을 통해 90% 이상으로 높여보자:

```
>>> 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([0.89707059, 0.8960948 , 0.90693604])
```

Error Analysis

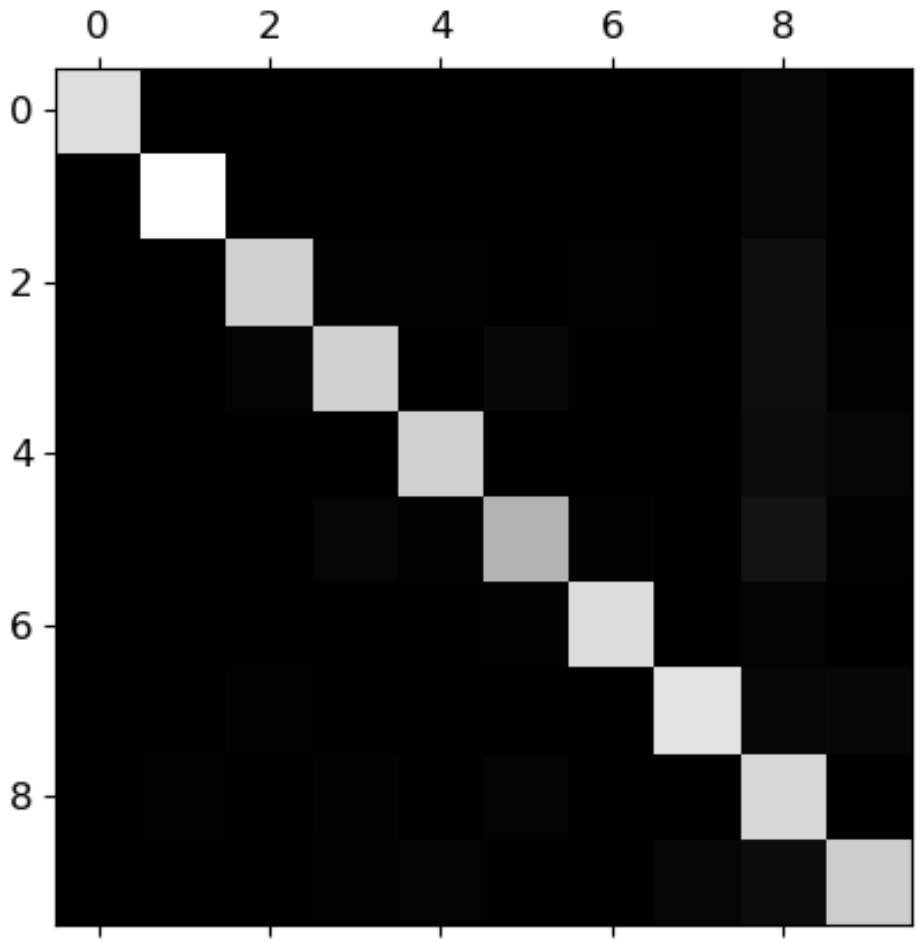
Promising 모델 발견 => 개선시키려면

- 에러 타입 분석

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([[5578,    0,   22,    7,    8,   45,   35,    5,  222,    1],
       [   0, 6410,   35,   26,    4,   44,    4,    8,  198,   13],
       [  28,   27, 5232,  100,   74,   27,   68,   37,  354,   11],
       [  23,   18,  115, 5254,    2,  209,   26,   38,  373,   73],
       [  11,   14,   45,   12, 5219,   11,   33,   26,  299,  172],
       [  26,   16,   31,  173,   54, 4484,   76,   14,  482,   65],
       [  31,   17,   45,    2,   42,   98, 5556,    3,  123,    1],
       [  20,   10,   53,   27,   50,   13,    3, 5696,  173,  220],
       [  17,   64,   47,   91,    3,  125,   24,   11, 5421,   48],
       [  24,   18,   29,   67,  116,   39,    1,  174,  329, 5152]])
```

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

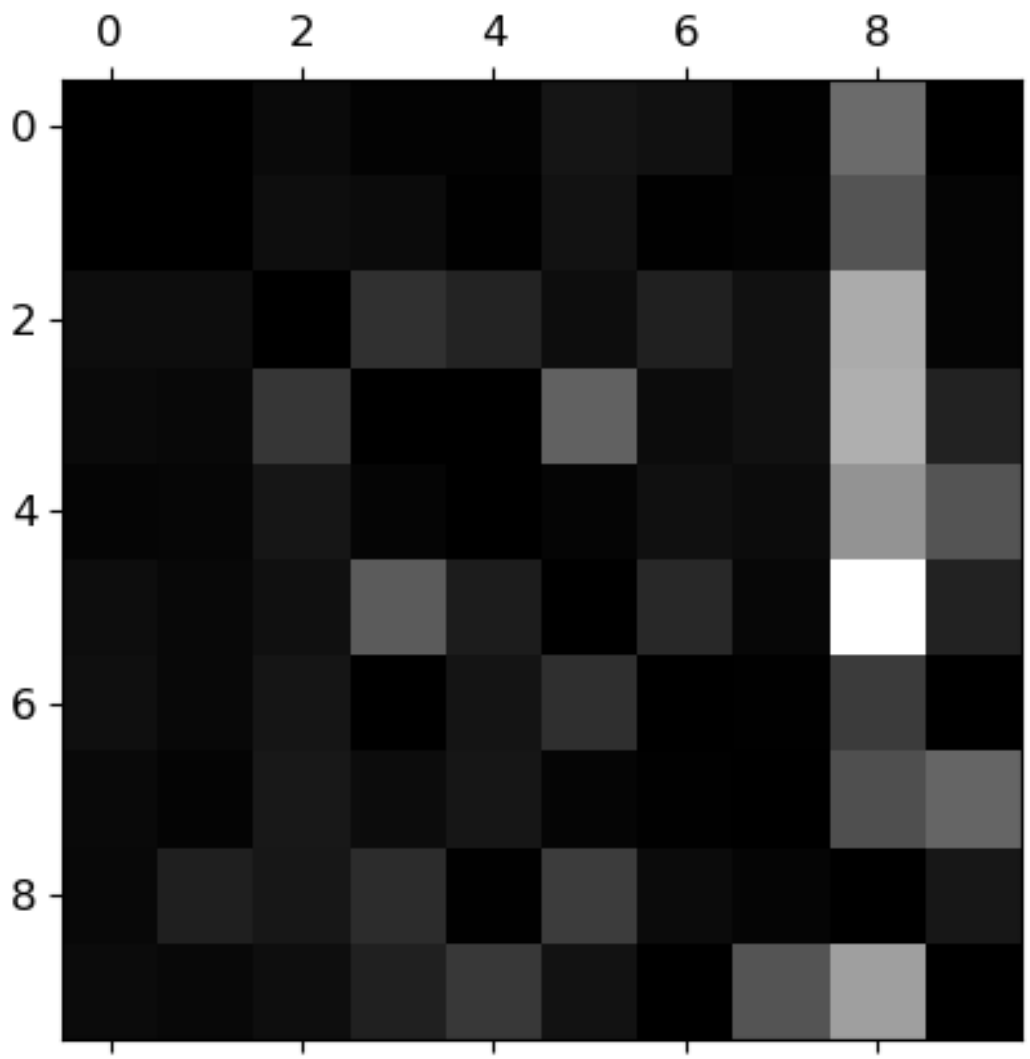


대부분 이미지 => 대각선 상에 있다!

5 => 다른 digit에 비해 약간 더 어두운 이유 추측:

- Dataset에 5 이미지가 더 적거나
- Classifier가 5를 잘 분류하지 못하거나 등

```
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()
```



Confusion matrix의 각 값을 해당 클래스에 있는 이미지의 개수로 나눔

⇒ 절대적인 에러 개수 대신 에러율로 비교!

오른쪽 그림:

8과 9 열이 매우 밝음 => 많은 이미지가 잘못 분류되었음을 의미

8과 9 행도 매우 밝음 => 많은 이미지가 잘못 분류되었음을 의미

실습과제 4-1

본문에 나오는 전체 내용 PyCharm에서 실행하기

참고: 제 04강 실습과제 #4 Classification - MNIST.pdf