로지스틱 회귀의 심화

- **01** 다중클래스 분류와 소프트맥스 분류
  02 다중클래스 분류를 코드로 구현하기
  03 ROC 커브와 AUC

학습목표

1. 다중클래스 분류와 소프트맥스 분류
2. 다중크래스 분류를 코드로 구현
3. ROC 커브와 AUC에 대해 알아본다.

01
다중클래스 분류와 소프트맥스 분류

1. 다중클래스 분류의 개념

- 다중클래스 분류(multi-class classification) : 2개 이상의 클래스를 가진 y 값에 대한 분류

1.1 다중클래스와 다중레이블

표 10-1 분류 작업에서 다중클래스와 다중레이블의 차이점

분류	다중클래스(multi-class) 분류	다중레이블(multi-label) 분류
작업	2개 이상의 클래스를 가진 분류 작업	상호 배타적이지 않은 속성 예측
중복 선택	중복 선택 불가능 →[1 0 0] 가능, [1 1 0] 불가	중복 선택 가능 →[1 1 0] 가능
예	과일 사진 분류 : 오렌지, 사과, 배	신문기사 분류 : 운동선수-연예인 결혼 기사 → 스포츠/연예 면

1.1 분류 접근

- One-vs-All : m개의 클래스가 존재할 때 각 클래스마다 분류기(classifier)를 생성하여 분류
  - One-vs-Rest라고도 부름
  - 대표적으로 소프트맥스 분류(softmax classification)
- One-vs-One : m개의 클래스가 있다면, 이 클래스의 분류기를 하나의 클래스로 하고 나머지 클래스의 분류기들을 만들어 최종적으로 각 분류기들의 결과를 투표로 결정
  - 총 $\frac{m(m-1)}{2}$ 개만큼의 분류기를 생성
  - 분류기가 많아질수록 정확도 높아지지만 비용도 증가

2. 소프트맥스 분류

2.1 소프트맥스 함수

- 시그모이드 함수로 다중클래스 분류 문제 다룰 수 있음
  - 각각의 클래스에 속하는지 속하지 않는지 이진분류기 m개를 생성한 후, 가장 높은 확률이 나오는 클래스를 선택
  - 분류기 번호 m에 대해 $h_m(x;\theta)$ 로 표현
  - 그러나 $h_m(x;\theta)$ 확률의 합이 1 이상이 된다는 문제 발생
  - 문제 해결 방법은 모든 클래스들의 발생 확률을 1로 정규화

01 다중클래스 분류와 소프트맥스 분류

- 소프트맥스 함수(softmax function) : 다중클래스 분류에서 여러 선형회귀의 출력 결과를 정규화하여 합이 1이 되도록 만드는 함수

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad \text{for } j = 1, 2, 3, \cdots, K$$

표 10-2 소프트맥스 함수 값 정리

z_j	e^{z_j}	$\frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$
2	7.389	0.609
1	2.718	0.224
–1	0.367	0.030
0.5	1.648	0.135

$$\sum_{j=1}^{K} \sigma(z)_j = \sum_{j=1}^{K} P_j = 1$$

01 다중클래스 분류와 소프트맥스 분류

In [1]:
```
import numpy as np

def softmax(values):
    array_values = np.exp(values)
    return array_values / np.sum(array_values)

values = [2, 1, 5, 0.5]
y = softmax(values) # array([0.04613281, 0.01697131,
0.92660226, 0.01029362])
y.sum()
```

Out [1]:
```
1.0
```

2.2 소프트맥스 함수로 구현하는 소프트맥스 분류

- 오즈비에 logit 함수를 붙여 최종적으로 구한 가중치 값

$$\frac{P_j}{1-P_j} \Rightarrow logit(P_j) = \log_e\left(\frac{P_j}{1-P_j}\right) = z = \theta^T X$$

- 기존의 오즈비는 이진분류이지만, 다중클래스 분류는 j 번째 대상에 대한 전체 대비 비율을 나타냄

$$\frac{P_j}{P_K} \Rightarrow \log \ddot{y}(P_j) = \log_e\left(\frac{P_j}{P_K}\right) = z_j = X^T\theta_j$$

$$\frac{P_j}{P_K} = e^{z_j}$$

01 다중클래스 분류와 소프트맥스 분류

- 다중클래스 분류에서는 j개의 클래스가 있다면 클래스의 개수만큼 가중치에 대한 벡터를 구함
- 피쳐 벡터와 각 클래스별로 존재하는 가중치 행렬의 선형결합을 z로 나타냄

$$\sum_{j=1}^{K} \frac{P_j}{P_K} = \sum_{j=1}^{K} e^{z_j} \Rightarrow \frac{1}{P_K} \sum_{j=1}^{K} P_j = \sum_{j=1}^{K} e^{z_j} \Rightarrow$$

$$\frac{1}{P_K} \times 1 = \sum_{j=1}^{K} e^{z_j} \Rightarrow \left(\because \sum_{j=1}^{K} P_j = 1 \right)$$

$$P_k = \frac{1}{\sum_{j=1}^{K} e^{z_j}}$$

$$\frac{P_j}{P_k} = e^{z_j} \Rightarrow \frac{P_j}{\sum_{j=1}^{K} e^{z_j}} = e^{z_j} \Rightarrow \because P_K = \frac{1}{\sum_{j=1}^{K} e^{z_j}}$$

$$P_j = \frac{e^{z_j}}{\sum_{j=1}^{K} e^{z_j}}$$

$$P_j = \frac{e^{z_j}}{\sum_{j=1}^{K} e^{z_j}} = \frac{e^{z_j}}{\sum_{j=1}^{K} e^{x^T \theta_j}} \qquad \because z = X^T \theta_j$$

$$\theta = \begin{bmatrix} \theta_1 \\ \theta_2 \\ \vdots \\ \theta_3 \end{bmatrix} = \begin{bmatrix} w_{10} & w_{11} & \cdots & w_{1i} \\ w_{20} & w_{21} & \cdots & w_{2i} \\ \vdots & \vdots & \ddots & \vdots \\ w_{j0} & w_{j1} & \cdots & w_{ji} \end{bmatrix}$$

3. 소프트맥스 함수로 학습하기

$$P_j = \frac{e^{x^T\theta_j}}{\sum_{j=1}^{K} e^{X^T\theta_j}}$$

- 오른쪽 수식에서 θ를 학습
  즉 각 클래스마다 적절한 θ_j를 찾기
- 각 가설함수는 각 클래스와 발생확률로 표현 가능

$$h_\theta(x) = \begin{bmatrix} P(y=1 \mid x;\theta) \\ P(y=2 \mid x;\theta) \\ \vdots \\ P(y=K \mid x;\theta) \end{bmatrix} = \frac{1}{\sum_{j=1}^{K} \exp(\theta^{(j)T} x)} \begin{bmatrix} \exp(\theta^{(1)T} x) \\ \exp(\theta^{(2)T} x) \\ \vdots \\ \exp(\theta^{(K)T} x) \end{bmatrix}$$

- 최대우도추정법(Maximum Likelihood Estation, MLE)을 사용해서 P_j 확률을 최대화하는 θ를 찾기

$$\arg\max_\theta \prod_{i=1}^{m} P(y^{(i)} \mid x^{(i)};\theta)$$

- 위 수식을 손실(loss)로 생각하여 수식 L로 표현하고 해당 값을 최대화하는 방향으로 정리

$$L = \prod_{i=1}^{m} P(y^{(i)} | x^{(i)}; \theta) = \prod_{i=1}^{m} \prod_{j=1}^{K} p_j^{(i)^{v_{ij}}} \Rightarrow$$

$$-\log L = -\log \prod_{i=1}^{m} \prod_{j=1}^{K} p_j^{(i)^{v_{ij}}} = -\sum_{i=1}^{m} \sum_{j=1}^{K} v_{ij} \log p_j^{(i)}$$

$$\text{where } v_{ij} \begin{cases} 1 & \text{if } y^{(i)} \text{is label } j \\ 0 & \text{if } y^{(i)} \text{is NOT label } j \end{cases}$$

$$l = -\sum_{i=1}^{m} \sum_{j=1}^{K} v_{ij} \log p_j^{(i)} \qquad where p_j^{(i)} = \frac{e^{z_j^{(i)}}}{\sum_{j=1}^{K} e^{z_j^{(i)}}}$$

- 최종적으로 $\frac{\partial l}{\partial z} \frac{\partial z}{\partial \theta}$ 을 구하고자 함

02
다중클래스 분류를 코드로 구현하기

1. mnist 데이터셋의 이해

- 손글씨를 숫자로 인식하는 이미지 분류 문제

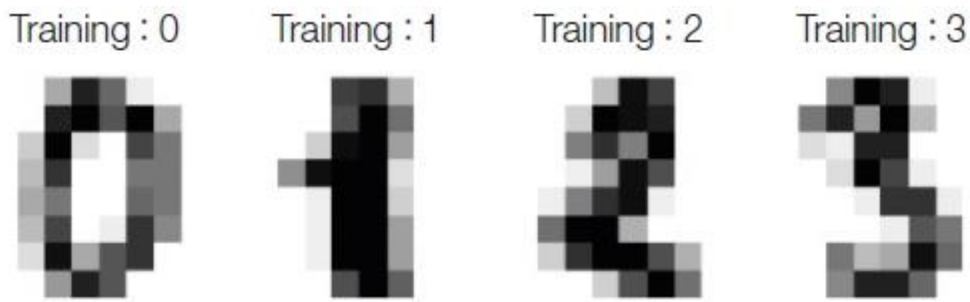


그림 10-1 사이킷런의 mnist 데이터셋 예제

- 컴퓨터는 이미지를 일종의 숫자로 변환하여 인식
  - 이미지를 일종의 점(dot)으로 생각하면 m×n만큼의 공간이 존재하고, 그 공간 안에서 색깔이 진할수록 높은 값, 색깔이 옅을수록 낮은 값을 가짐

2. 데이터 불러오기

- datasets 모듈을 호출
- load_digits 함수로 딕셔너리 타입 데이터를 불러온다

In [1]:
```
from sklearn import datasets
digit_dataset = datasets.load_digits()
digit_dataset.keys()
```

Out [1]:
```
dict_keys(['data', 'target', 'frame',
'feature_names', 'target_names', 'images', 'DESCR'])
```

02 다중클래스 분류를 코드로 구현하기

```
In [2]:  digit_dataset["images"].shape
Out [2]: (1797, 8, 8)
In [3]:  digit_dataset["target"][0]
Out [3]: 0
In [4]:  digit_dataset["images"][0]
Out [4]: array([[ 0., 0., 5., 13., 9., 1., 0., 0.],
                [ 0., 0., 13., 15., 10., 15., 5., 0.],
                [ 0., 3., 15., 2., 0., 11., 8., 0.],
                [ 0., 4., 12., 0., 0., 8., 8., 0.],
                [ 0., 5., 8., 0., 0., 9., 8., 0.],
                [ 0., 4., 11., 0., 1., 12., 7., 0.],
                [ 0., 2., 14., 5., 10., 12., 0., 0.],
                [ 0., 0., 6., 13., 10., 0., 0., 0.]])
```

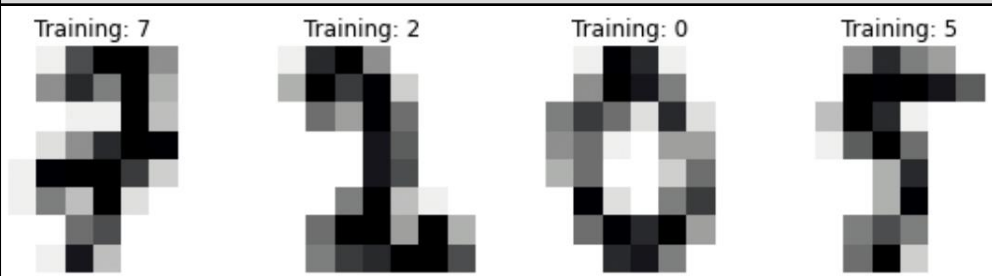
02 다중클래스 분류를 코드로 구현하기

In [5]:

```
import matplotlib.pyplot as plt
from random import randint
_, axes = plt.subplots(nrows=1, ncols=4, figsize=(10, 3))
# (1)

for ax in axes: # (2)
    num = randint(1, 1000) # (3)
    image = digit_dataset["images"][num]
    label = digit_dataset["target"][num]
    ax.set_axis_off()
    ax.imshow(image, cmap=plt.cm.gray_r,
interpolation='nearest') # (4)
    ax.set_title('Training: %i' % label)
```

Out [5]:



[TIP] 결과값에 색을 지정하는 요소(property)인 plt.cm.gray_r을 변경하면 좀 더 다양한 형태로 값 표현이 가능하다

- 데이터가 8×8 행렬이므로 2D 이미지로 표현되었지만 다음 코드와 같이 총 64개의 피쳐(feature)를 가진 하나의 데이터로 받을 수 있음

In [6]:
```
digit_dataset["data"][0].shape
```
Out [6]:
```
(64,)
```

3. 데이터 분류하기

- 데이터를 훈련 데이터셋과 테스트 데이터셋으로 구분

In [7]:
```
from sklearn.model_selection import train_test_split

X = digit_dataset["data"] # (1)
y = digit_dataset["target"] # (1)
X_train, X_test, y_train, y_test = train_test_split(X, y)
# (2)
```

4. 모델 생성하기

- ovr : 클래스 모드를 모두 이진모델로 만들어 학습
- multinomial : 소프트맥스 함수를 사용하여 계산하는 방식. 경사하강법의 매개변수 solver를 sag으로 변경

In [8]:
```
from sklearn.linear_model import LogisticRegression

logreg_ovr = LogisticRegression(multi_class="ovr")
logreg_softmax =
LogisticRegression(multi_class="multinomial",
solver="sag")

logreg_ovr.fit(X_train, y_train)
logreg_softmax.fit(X_train, y_train)
```

Out [8]:
```
LogisticRegression(multi_class='multinomial',
solver='sag')
```

5. 성능 측정하기

- 일반적으로 다중클래스 분류도 기존 혼동행렬을 사용
- 각 클래스 대비 예측한 값을 행렬 형태로 표현

In [9]:
```
from sklearn.metrics import confusion_matrix
y_pred = logreg_ovr.predict(X_test).copy()
y_true = y_test.copy()
confusion_matrix(y_true, y_pred)
```

Out [9]:
```
array([[47, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [ 0, 49, 1, 0, 0, 0, 0, 0, 0, 0],
       [ 0, 0, 49, 2, 0, 0, 0, 0, 0, 0],
       [ 0, 0, 0, 37, 0, 1, 0, 0, 1, 0],
       [ 0, 0, 0, 0, 41, 0, 0, 0, 0, 0],
       [ 0, 1, 0, 0, 1, 41, 1, 0, 0, 1],
       [ 0, 0, 0, 0, 2, 0, 36, 0, 0, 0],
       [ 0, 0, 0, 0, 1, 0, 0, 41, 0, 1],
       [ 0, 0, 0, 0, 0, 0, 0, 0, 44, 0],
       [ 1, 0, 0, 0, 0, 0, 0, 0, 1, 50]], dtype=int64)
```

02 다중클래스 분류를 코드로 구현하기

- 라벨별로 분류 성능을 수치화하여 표시

In [10]:
```python
from sklearn.metrics import classification_report
print(classification_report(y_true, y_pred))
```

Out [10]:
```
              precision    recall  f1-score   support
           0       0.98      1.00      0.99        47
           1       0.98      0.98      0.98        50
           2       0.98      0.96      0.97        51
           3       0.95      0.95      0.95        39
           4       0.91      1.00      0.95        41
           5       0.98      0.91      0.94        45
           6       0.97      0.95      0.96        38
           7       1.00      0.95      0.98        43
           8       0.96      1.00      0.98        44
           9       0.96      0.96      0.96        52

    accuracy                           0.97       450
   macro avg       0.97      0.97      0.97       450
weighted avg       0.97      0.97      0.97       450
```

02 다중클래스 분류를 코드로 구현하기

- micro를 선택하면 전체 평균값, macro를 선택하면 각 라벨별 결과의 합에 대한 평균을 나타냄

```
In [11]: result = confusion_matrix(y_true, y_pred)
         result.diagonal().sum() / result.sum(axis=0).sum()
Out [11]: 0.9533333333333333
```

```
In [12]: from sklearn.metrics import precision_score
         precision_score(y_true, y_pred, average="micro")
Out [12]: 0.9666666666666667
In [13]: precision_score(y_true, y_pred, average="macro")
Out [13]: 0.9666219376328072
In [14]: precision_score(y_true, y_pred, average=None)
Out [14]: array([0.97916667, 0.98      , 0.98      ,
                 0.94871795, 0.91111111, 0.97619048,
                 0.97297297, 1.        , 0.95652174,
                 0.96153846])
```

03
ROC 커브와 AUC

03 ROC 커브와 AUC

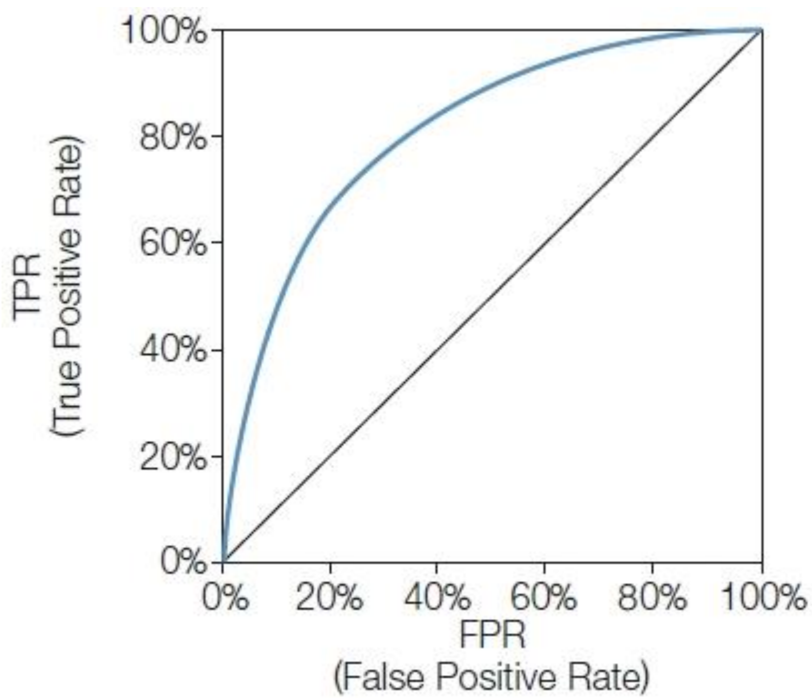
1. 정밀도와 민감도의 트레이드오프

- 정밀도(precision)와 민감도(recall)는 일반적으로 둘 다 동시에 상승하기 어렵고 임계값(threshold)에 따라 변화가 일어남
  - 두 값을 모두 고려하여 성능을 측정하기 쉽지 않음
- ROC 커브(ROC curve) : 분류기의 임계값을 지속적으로 조정하여 정밀도와 민감도 간의 비율을 도식화
  - ‘Receiver Operating Characteristics’의 약자
  - 클래스의 예측 확률이 나오는 모델에 사용 가능

03 ROC 커브와 AUC

2. ROC 커브 표현하기

- TPR(True Positive Rate)과 FPR(False Positive Rate)을 각각 y축, x축에 나타내어 그래프를 작성



데이터	클래스	임계값
1	P	0.9
2	P	0.8
3	N	0.7
4	P	0.6
5	P	0.55
6	N	0.54
7	N	0.53
8	N	0.51
9	P	0.5
10	N	0.4

그림 10-2 일반적인 ROC 커브와 각 데이터의 예측 확률

03 ROC 커브와 AUC

표 10-3 데이터 정리

데이터	클래스	임계값	TP	FP	TN	FN	TPR	FPR
1	P	0.9	1	0	5	4	0.2	0
2	P	0.8	2	0	5	3	0.4	0
3	N	0.7	2	1	4	3	0.4	0.2
4	P	0.6	3	1	4	2	0.6	0.2
5	P	0.55	4	1	4	1	0.8	0.2
6	N	0.54	4	2	3	1	0.8	0.4
7	N	0.53	4	3	2	1	0.8	0.6
8	N	0.51	4	4	1	1	0.8	0.8
9	P	0.5	5	4	0	1	1	1
10	N	0.4	5	5	0	0	1	1

03 ROC 커브와 AUC

- TPR과 FPR의 값을 연결하여 그래프를 작성

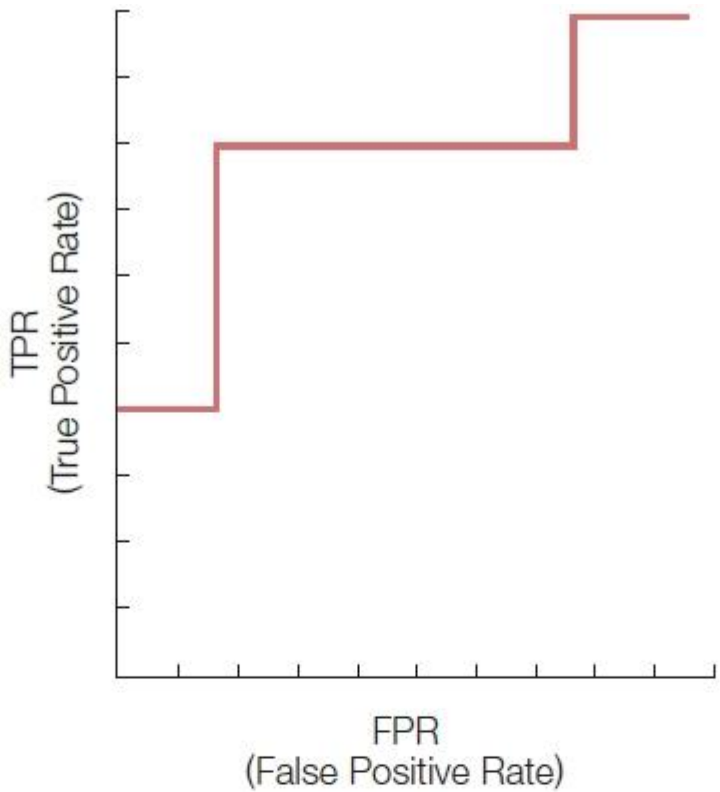


그림 10-3 [표 10-3]을 기준으로 한 ROC 커브

03 ROC 커브와 AUC

- AUC(Area Under Curve) : ROC 커브 하단의 넓이

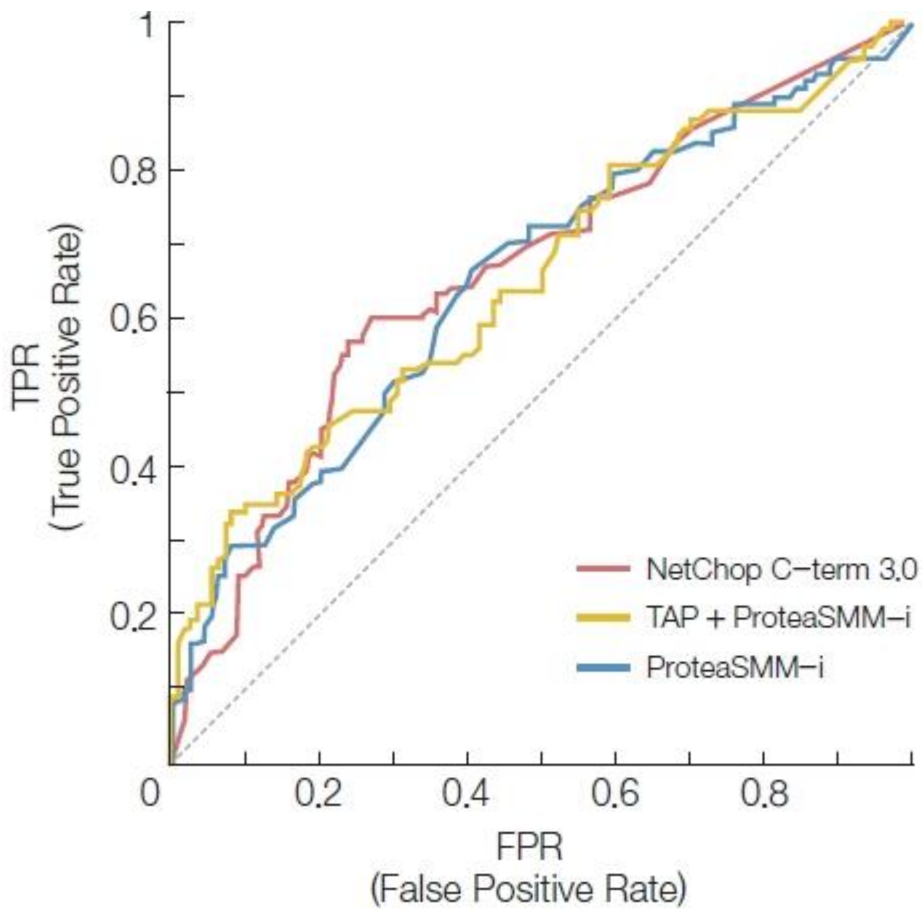


그림 10-4 여러 모델의 성능을 상대적으로 비교하기 위한 그래프

- 대표적인 ROC 커브로 모델들의 성능을 단 하나의 숫자로 표현할 수 있다는 점에서 불균형 데이터셋(imbalanced dataset)의 성능을 평가할 때 많이 사용

03 ROC 커브와 AUC

3. ROC 커브와 AUC를 사이킷런 코드로 구현하기

- 정답 y 값과 각 항목별 예측 확률을 scores에 저장
- ROC 커브 함수인 roc_curve로 fpr, tpr, thresholds 반환

In [1]:
```
import numpy as np
from sklearn import metrics

y = np.array([1, 1, 2, 2])
scores = np.array([0.1, 0.4, 0.35, 0.8])
fpr, tpr, thresholds = metrics.roc_curve(y, scores,
pos_label=2)

# fpr - array([0. , 0. , 0.5, 0.5, 1. ])
# tpr - array([0. , 0.5, 0.5, 1. , 1. ])
# thresholds - array([1.8 , 0.8 , 0.4 , 0.35, 0.1 ])
```

03 ROC 커브와 AUC

In [2]:
```
y = np.array([1, 1, 2, 2])
pred = np.array([0.1, 0.4, 0.35, 0.8])
fpr, tpr, thresholds = metrics.roc_curve(y, pred,
pos_label=2)
roc_auc = metrics.auc(fpr, tpr)
roc_auc
```

Out [2]:
```
0.75
```

In [3]:
```
import matplotlib.pyplot as plt

plt.figure()
lw = 2
plt.plot(fpr, tpr,
         lw=lw, label='ROC curve (area = %0.2f)' %
roc_auc)
```

03 ROC 커브와 AUC

```
plt.plot([0, 1], [0, 1], color='navy', lw=lw,
linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic example')
plt.legend(loc="lower right")
plt.show()
```

Out [3]: