평가지표 알아보기 - ROC 커브, AUC

1.1.1 이진 분류의 평가지표

1.1.2 임계값과 평가지표

1.1.3 평가지표 - ROC 커브, AUC

1.1.4 다중 분류의 평가지표

학습 내용

- 정밀도 재현율을 구하는 함수를 알아본다.
- 재현율(recall, Tprate), Fprate를 구하는 함수를 알아본다.
- 정밀도(precision), 재현율 그래프를 확인해 본다.
- Tprate, Fprate의 그래프를 확인해 본다.(ROC 커브)
- ROC커브를 구하고 AUC에 대해 이해해본다.

목차

한글 설정

In [81]:

```
import matplotlib
from matplotlib import font_manager, rc
import platform
import warnings
warnings.filterwarnings(action='ignore')
```

In [82]:

```
### 한글
path = "C:/Windows/Fonts/malgun.ttf"
if platform.system() == "Windows":
    font_name = font_manager.FontProperties(fname=path).get_name()
    rc('font', family=font_name)
elif platform.system()=="Darwin":
    rc('font', family='AppleGothic')
else:
    print("Unknown System")


matplotlib.rcParams['axes.unicode_minus'] = False
```

01. 데이터 준비 및 라이브러리 임포트

목차로 이동하기

정밀도 재현율 곡선을 이용하여 성능을 판단해 보기

In [83]:

```
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import numpy as np
from sklearn.metrics import classification_report
```

In [84]:

```
from mglearn.datasets import make_blobs
X, y = make_blobs(n_samples=(400, 50),
                  centers=2, cluster_std=[7.0, 2],
                  random_state=22)

print(X.shape, y.shape)
```

```
(450, 2) (450,)
```

In [85]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
svc = SVC(gamma=.05).fit(X_train, y_train)
tree = DecisionTreeClassifier().fit(X_train, y_train)
```

In [86]:

```
# SVC 모델
pred = svc.predict(X_test)
print(classification_report(y_test, pred))
```

```
              precision    recall  f1-score   support

           0       0.97      0.89      0.93       104
           1       0.35      0.67      0.46         9

    accuracy                           0.88       113
   macro avg       0.66      0.78      0.70       113
weighted avg       0.92      0.88      0.89       113
```

In [87]:

```
# 의사결정 트리 모델
pred = tree.predict(X_test)
print(classification_report(y_test, pred))
```

```
              precision    recall  f1-score   support

           0       0.98      0.88      0.92       104
           1       0.35      0.78      0.48         9

    accuracy                           0.87       113
   macro avg       0.66      0.83      0.70       113
weighted avg       0.93      0.87      0.89       113
```

02. 정밀도-재현율 곡선 확인

목차로 이동하기

- 새로운 모델을 만들 때, 운영을 위해 90% 재현율을 유지하는 것등의 필요조건을 지정할 때가 있다.
  - 이를 운영 포인트를 지정한다고 말하고, 운영 포인트를 고정하면 비즈니스 목표 설정이나 고객내 다른 그룹의 성능을 보장하는데 도움이 된다.
- 이런 경우에 과제를 잘 이해하기 위해 모든 임계값을 조사하거나, 정밀도나 재현율의 모든 장단점을 살펴본다.
  - 이를 위해 정밀도-재현율 곡선을 사용

정밀도(x)와 재현율(y) - ROC 커브 확인해 보기

- precision_recall_curve() 메서드 이용

In [88]:

```
from sklearn.metrics import precision_recall_curve

precision, recall, thresholds = precision_recall_curve(y_test,
                                                       svc.decision_function(X_test))
```

In [89]:

```
# 부드러운 곡선을 위해 데이터 포인트 수를 늘립니다
X, y = make_blobs(n_samples=(4000, 500),
                  centers=2,
                  cluster_std=[7.0, 2],
                  random_state=22)

print(X.shape, y.shape)


X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=10)
```

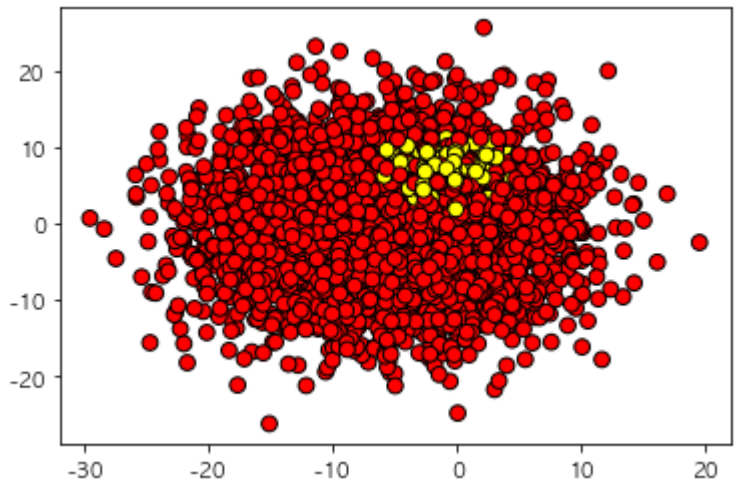
```
(4500, 2) (4500,)
```

In [90]:

```
plt.scatter(X[:,0], X[:,1],
            c=y,
            cmap=plt.cm.autumn,  # plt.cm.Blues, RdYlGn, BrBG, Greens, RdGy, YlOrRd,
            s=60, edgecolors='k')
```

Out[90]:

```
<matplotlib.collections.PathCollection at 0x7fa7e3aa5eb0>
```



정밀도-재현율 곡선 확인

In [91]:

```
svc = SVC(gamma=.05).fit(X_train, y_train)

pred = svc.decision_function(X_test)  #  0의 값을 기준으로 분포
print(pred[0:10])
```

```
[-1.09425577 -1.10667545 -1.10736997 -1.19140534 -1.22918652 -1.197499
83
 -1.17059048 -1.24720063 -1.19551381 -1.11214569]
```

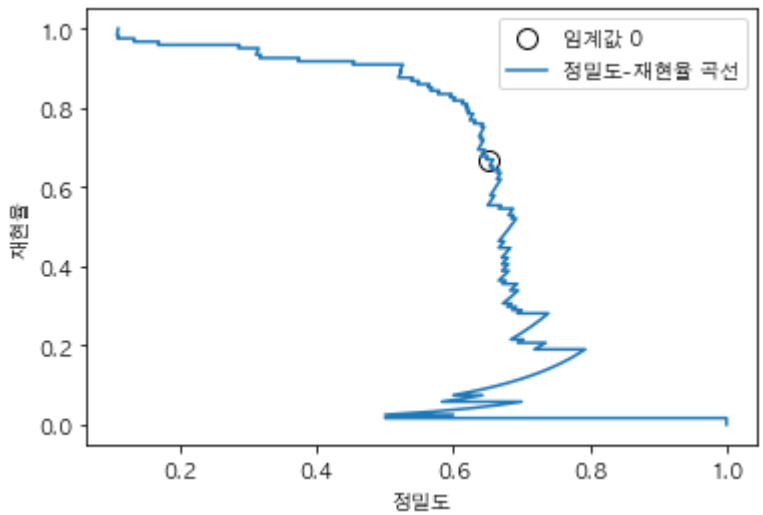
In [92]:

```
precision, recall, thresholds = precision_recall_curve(y_test, pred)

print("임계값 : ", thresholds.min(),  thresholds.max())

# 0에 가까운 임계값을 찾습니다
close_zero = np.argmin(np.abs(thresholds))  # thresholds의 절대값이 가장 작은 것(위치)
print(close_zero)

plt.plot(precision[close_zero],
         recall[close_zero], 'o',
         markersize=10,
         label="임계값 0",
         fillstyle="none", c='k')

plt.plot(precision, recall, label="정밀도-재현율 곡선")
plt.xlabel("정밀도")
plt.ylabel("재현율")
plt.legend(loc="best")
```

```
임계값 :  -1.5528391475651482 1.4968799824567065
983
```

Out[92]:

```
<matplotlib.legend.Legend at 0x7fa7e36bcbb0>
```



정밀도

$$\text{정밀도(precision)} = \frac{\text{잘 예측(TP)}}{\text{예측을 양성으로 한 것 전체(TP+FP)}}$$

재현율(recall, 민감도, TPR)

- 실제 양성 데이터를 양성으로 잘 예측

$$\text{민감도(recall, 재현율)} = \frac{\text{잘 예측(TP)}}{\text{실제 값이 양성인것 전체(TP+FN)}}$$

가짜 양성 비율(Fprate)

- 1 - 특이도
- 실제 음성 데이터 중에 음성(0) 데이터를 잘못 예측한 비율

$$\text{가짜 양성 비율(Fprate)} = \frac{\text{틀린 예측(FP)}}{\text{실제 값이 음성인것 전체(FP + TN)}}$$

03. 정밀도-재현율의 커브(랜덤 포레스트 모델)

목차로 이동하기

In [93]:

```
X_train.shape  # 현재 feature 개수
```

Out[93]:

```
(3375, 2)
```

In [94]:

```
from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier(n_estimators=100, random_state=0, max_features=2)
rf.fit(X_train, y_train)
pred =  rf.predict_proba(X_test)[:, 1]
pred
```

Out[94]:

```
array([0.  , 0.  , 0.  , ..., 0.  , 0.66, 0.  ])
```

SVC모델과 RandomForest모델의 비교

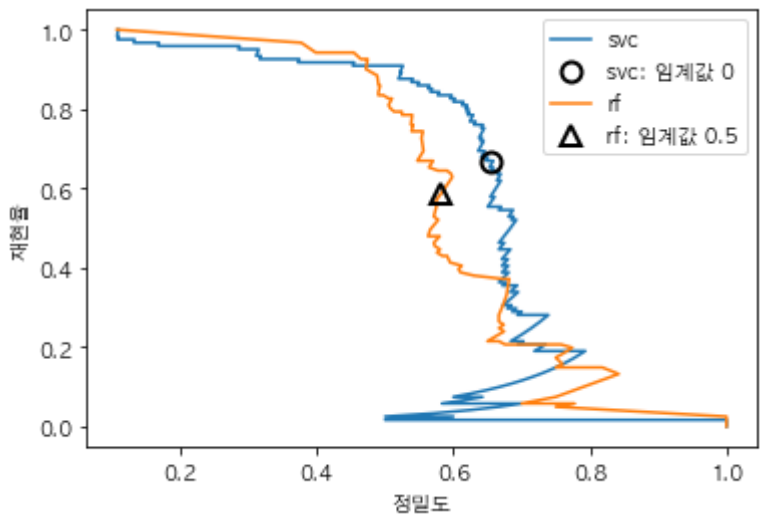
In [95]:

```
# SVC모델 그래프
plt.plot(precision, recall, label="svc")

plt.plot(precision[close_zero],
         recall[close_zero], 'o',
         markersize=10,
         label="svc: 임계값 0",
         fillstyle="none",
         c='k',
         mew=2)

# RandomForestClassifier는 decision_function 대신 predict_proba를 제공합니다.
precision_rf, recall_rf, thresholds_rf = precision_recall_curve(y_test, pred)

# 랜덤포레스트 그래프
plt.plot(precision_rf, recall_rf, label="rf")

close_zero_rf = np.argmin(  np.abs(thresholds_rf - 0.5)  )    # 임계값이 0.5 위치
print(close_zero_rf)

plt.plot(precision_rf[close_zero_rf], recall_rf[close_zero_rf], '^', c='k',
         markersize=10, label="rf: 임계값 0.5", fillstyle="none", mew=2)
plt.xlabel("정밀도")
plt.ylabel("재현율")
plt.legend(loc="best")
```

45

Out[95]:

<matplotlib.legend.Legend at 0x7fa7e3d5e580>



- [이해하기] 정밀도 재현율 곡선이 오른쪽 위로 갈수록 더 나은 분류기를 나타낸다.
- 재현율이 매우 높거나, 정밀도가 매우 높을 때는 랜덤포레스트가 더 좋은 성능을 갖는다.
- 정밀도 0.7부분에서는 SVM이 좋음
- f1-score 점수만으로 모델의 성능을 평가한다면 이런 세세한 부분을 놓칠 수 있다.
    - f1-score는 정밀도-재현율 곡선의 **한 지점인 기본 임계값에 대한 점수**이기 때문에

(실습) 의사결정트리 모델 추가해 보기

04. 평균 정밀도 확인

목차로 이동하기

In [96]:

```
from sklearn.metrics import f1_score

rf_f1score = f1_score(y_test, rf.predict(X_test) )
svc_f1score = f1_score(y_test, svc.predict(X_test))

print("랜덤 포레스트의 f1_score: {:.3f}".format(rf_f1score))
print("svc의 f1_score: {:.3f}".format(svc_f1score))
```

```
랜덤 포레스트의 f1_score: 0.573
svc의 f1_score: 0.661
```

모델을 비교하기 위한 방법 중의 하나로 정밀도-재현율 곡선의 아랫부분 면적을 이용

- 이 면적을 평균 정밀도(average precision)이라 한다.
- 함수로 average_precision_score함수로 **평균 정밀도 계산**이 가능하다.

In [97]:

```
from sklearn.metrics import average_precision_score

## 확률 예측
rf_pro = rf.predict_proba(X_test)[:, 1]
svc_dcfun = svc.decision_function(X_test)

ap_rf = average_precision_score(y_test, rf_pro)
ap_svc = average_precision_score(y_test, svc_dcfun)

print("랜덤 포레스트의 평균 정밀도: {:.3f}".format(ap_rf))
print("svc의 평균 정밀도: {:.3f}".format(ap_svc))
```

```
랜덤 포레스트의 평균 정밀도: 0.608
svc의 평균 정밀도: 0.632
```

- 평균 정밀도는 두 모델의 성능이 비슷하다. 위의 f1_score의 결과와는 조금 다르다.

05. ROC 곡선과 AUC 알아보기

목차로 이동하기

ROC 곡선

- **ROC 곡선은 여러 임계값에서 분류기의 특성을 분석하는데 널리 사용되는 도구.**
- ROC 곡선은 분류기의 모든 임계값을 고려
- 앞의 그래프의 x는 정밀도, y가 재현율(TPR)이었다면
  - ROC곡선은 x는 (False Positive rate), y를 재현율(True Positive rate)로 한것.
  - 진짜 양성 비율(TPR), 거짓 양성 비율(FPR)

ROC 와 AUC

- x축은 Fprate, y축은 Tprate(재현율, 민감도)로 한다.
- FPrate는 1-특이도와 같다
- FPrate는 실제 음성인 데이터 중에 양성으로 예측하여 틀린 것의 비율

$$\text{FPRate} = \frac{\text{틀린 예측(FP)}}{\text{실제 값이 음성인것 전체(FP + TN)}}$$

In [98]:

```
from sklearn.metrics import roc_curve
fpr, tpr, thresholds = roc_curve(y_test, svc.decision_function(X_test))
```

임계값에 따른 각각의 Fprate, Tprate를 구하기

In [99]:

```
fpr.shape, tpr.shape, thresholds
```

Out[99]:

```
((120,),
 (120,),
 array([ 2.49687998,  1.49687998,  1.47581867,  1.4610055 ,  1.4580211
3,
         1.44126949,  1.39147826,  1.37380751,  1.35378001,  1.3511580
9,
         1.23928869,  1.22953332,  1.21983036,  1.20198557,  1.1837964
3,
         1.16423601,  1.03010903,  0.96014974,  0.95503679,  0.9238038
,
         0.91992255,  0.91412964,  0.91086392,  0.91041837,  0.8667439
1,
         0.85989618,  0.85955523,  0.77050185,  0.76271236,  0.7603486
8,
         0.74948191,  0.74083693,  0.67295553,  0.6313474 ,  0.5961244
3,
         0.56820147,  0.5421393 ,  0.52460529,  0.51559663,  0.5071938
4,
         0.37819992,  0.35974527,  0.35286999,  0.35078072,  0.3363391
2,
         0.28457478,  0.28263032,  0.262315  ,  0.22681224,  0.2248964
2,
         0.20402619,  0.19841834,  0.1535657 ,  0.13174612,  0.1107180
6,
         0.06216232,  0.05418311,  0.05175504,  0.02982469, -0.0108257
9,
        -0.0441958 , -0.07263297, -0.14957616, -0.16204013, -0.2231269
8,
        -0.26617542, -0.28148442, -0.29640122, -0.33187486, -0.3636459
1,
        -0.37896216, -0.413147  , -0.41411063, -0.46562446, -0.4780611
6,
        -0.48767225, -0.5058179 , -0.51036328, -0.5110437 , -0.5264417
5,
        -0.53222803, -0.58481903, -0.58940514, -0.630053  , -0.6568114
1,
        -0.69947483, -0.70880328, -0.76236455, -0.76766451, -0.7913357
7,
        -0.80125093, -0.84469204, -0.84626488, -0.89704645, -0.8998519
7,
        -0.91717414, -0.93386616, -0.98339756, -0.98997147, -0.9919836
2,
        -0.99232122, -1.00442937, -1.00450048, -1.0275707 , -1.0276451
6,
        -1.05723133, -1.05811294, -1.06372408, -1.06460692, -1.0912274
5,
        -1.0915653 , -1.11441756, -1.11452809, -1.18688593, -1.1868962
6,
        -1.38761101, -1.3947979 , -1.53375669, -1.55283915, -1.7798835
6]))
```

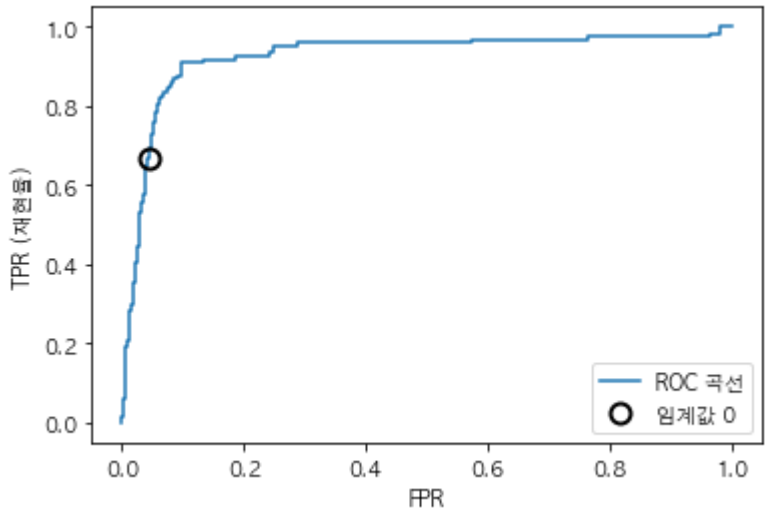
ROC곡선은 왼쪽 위쪽에 가까울 수록 성능이 좋은 것을 나타낸다.

In [100]:

```
plt.plot(fpr, tpr, label="ROC 곡선")
plt.xlabel("FPR")
plt.ylabel("TPR (재현율)")

# 임계값이 0 근처의 임계값을 찾습니다
close_zero = np.argmin(np.abs(thresholds))
plt.plot(fpr[close_zero], tpr[close_zero], 'o', markersize=10,
         label="임계값 0", fillstyle="none", c='k', mew=2)
plt.legend(loc=4)
```

Out[100]:

```
<matplotlib.legend.Legend at 0x7fa7e3fd5c40>
```

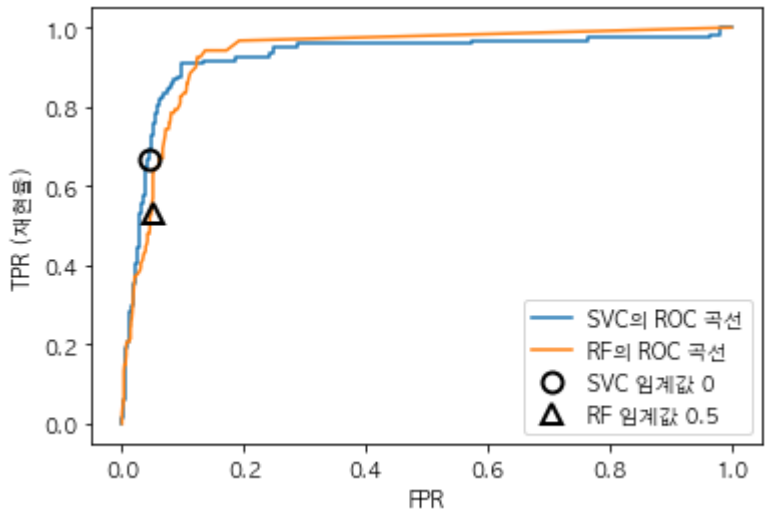


In [101]:

```
from sklearn.metrics import roc_curve
fpr_rf, tpr_rf, thresholds_rf = roc_curve(y_test, rf.predict_proba(X_test)[:, 1])

plt.plot(fpr, tpr, label="SVC의 ROC 곡선")
plt.plot(fpr_rf, tpr_rf, label="RF의 ROC 곡선")

plt.xlabel("FPR")
plt.ylabel("TPR (재현율)")
plt.plot(fpr[close_zero], tpr[close_zero], 'o', markersize=10,
         label="SVC 임계값 0", fillstyle="none", c='k', mew=2)

close_05_rf = np.argmin(np.abs(thresholds_rf - 0.5))
plt.plot(fpr_rf[close_05_rf], tpr[close_05_rf], '^', markersize=10,
         label="RF 임계값 0.5", fillstyle="none", c='k', mew=2)

plt.legend(loc=4)
```

Out[101]:

```
<matplotlib.legend.Legend at 0x7fa7e3ab8520>
```



In [102]:

```
from sklearn.metrics import roc_auc_score
rf_auc = roc_auc_score(y_test, rf.predict_proba(X_test)[:, 1])
svc_auc = roc_auc_score(y_test, svc.decision_function(X_test))

print("랜덤 포레스트의 AUC: {:.3f}".format(rf_auc))
print("SVC의 AUC: {:.3f}".format(svc_auc))
```

```
랜덤 포레스트의 AUC: 0.933
SVC의 AUC: 0.925
```

REF

- plt.cm.___ : https://chrisalbon.com/python/basics/set_the_color_of_a_matplotlib/ (https://chrisalbon.com/python/basics/set_the_color_of_a_matplotlib/)