

# Introduction to Machine Learning with Python

## 5. Model Evaluation and Improvement(2)

Honedae Machine Learning Study Epoch #2

# Contacts

Haesun Park

Email : [haesunrpark@gmail.com](mailto:haesunrpark@gmail.com)

Meetup: <https://www.meetup.com/Hongdae-Machine-Learning-Study/>

Facebook : <https://facebook.com/haesunrpark>

Blog : <https://tensorflow.blog>

# Book

파이썬 라이브러리를 활용한 머신러닝, 박해선.

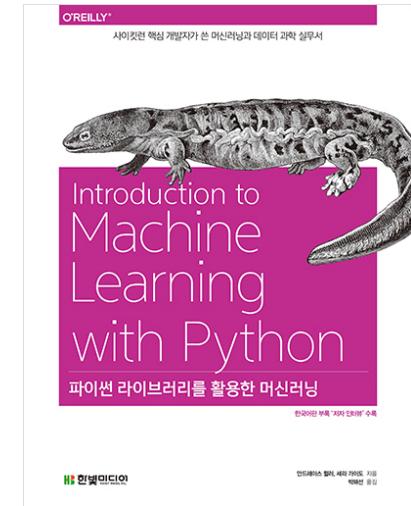
(Introduction to Machine Learning with Python, Andreas Muller & Sarah Guido의 번역서입니다.)

번역서의 1장과 2장은 블로그에서 무료로 읽을 수 있습니다.

원서에 대한 프리뷰를 온라인에서 볼 수 있습니다.

Github:

[https://github.com/rickiepark/introduction\\_to\\_ml\\_with\\_python/](https://github.com/rickiepark/introduction_to_ml_with_python/)



# 평가 지표와 측정

# 평가 지표

일반적으로 회귀:  $R^2$ , 분류: 정확도를 사용합니다.

비즈니스 임팩트를 고려하여 목표를 설정합니다(쇼핑몰의 방문자수/매출, 교통사고 횟수/입원 환자수 등)

비즈니스 지표를 얻으려면 운영 시스템에 적용해야 알 수 있는 경우가 많으므로 대리할 수 있는 평가 지표를 사용합니다(보행자 이미지를 사용한 자율 주행 테스트)

시스템의 목표에 따라 방문 고객이 10% 늘어나는 모델을 찾을 수 있지만 매출은 15% 줄어들 수 있습니다(경우에 따라 고객이 아니라 매출이 비즈니스 지표가 될 수 있습니다).

# 이진 분류

# 에러의 종류

양성 클래스(관심 대상)와 음성 클래스로 나뉩니다.

암진단: 암([양성 테스트](#), [악성](#)), 정상([음성 테스트](#))

[양성 테스트](#)(암)를 양성 클래스, [음성 테스트](#)(정상)을 음성 클래스라 할 때  
정상을 양성 클래스로 분류(거짓 양성 false positive): 추가적인 검사 동반  
암을 음성 클래스로 분류(거짓 음성 false negative): 건강을 해침

통계학에서는 거짓 양성을 타입 I 에러, 거짓 음성을 타입 II 에러라고도 합니다.

거짓 양성과 거짓 음성의 중요도가 비슷한 경우는 드물며 오류를 비용으로  
환산하여 비즈니스적인 판단을 해야 합니다(암진단의 경우는 거짓음성이 매우  
중요합니다).

# 불균형 데이터셋

예) 광고 노출 수와 클릭 수: 99 vs 1 (현실에서는 0.1% 미만입니다)

무조건 ‘클릭 아님’으로 예측하면 99%의 정확도를 가진 분류기가 됩니다.

불균형한 데이터셋에서는 정확도만으로는 모델이 진짜 좋은지  
모릅니다(현실에서는 불균형한 데이터셋이 많습니다).

```
In [41]: from sklearn.datasets import load_digits  
  
digits = load_digits()  
y = digits.target == 9  
  
X_train, X_test, y_train, y_test = train_test_split(  
    digits.data, y, random_state=0)
```

숫자 9는 1, 나머지는 0인 타깃값



# 더미 vs 결정 트리

```
In [42]: from sklearn.dummy import DummyClassifier  
dummy_majority = DummyClassifier(strategy='most_frequent').fit(X_train, y_train)  
pred_most_frequent = dummy_majority.predict(X_test)  
print("예측된 레이블의 고유값: {}".format(np.unique(pred_most_frequent)))  
print("테스트 점수: {:.2f}".format(dummy_majority.score(X_test, y_test)))
```

예측된 레이블의 고유값: [False]

테스트 점수: 0.90

무조건 9아님(0)으로 예측을 만듦

실제로는 검증세트를 사용해야 합니다

```
In [43]: from sklearn.tree import DecisionTreeClassifier  
tree = DecisionTreeClassifier(max_depth=2).fit(X_train, y_train)  
pred_tree = tree.predict(X_test)  
print("테스트 점수: {:.2f}".format(tree.score(X_test, y_test)))
```

테스트 점수: 0.92

불균형한 데이터셋에서는 모델의 정확도가 좋은 측정 방법이 아닙니다.

# 더미 vs 로지스틱 회귀

strategy 기본값: stratified  
클래스 비율(10%:1, 90%:0) 대로 랜덤하게 예측

```
In [44]: from sklearn.linear_model import LogisticRegression

dummy = DummyClassifier().fit(X_train, y_train)
pred_dummy = dummy.predict(X_test)
print("dummy 점수: {:.2f}".format(dummy.score(X_test, y_test)))

logreg = LogisticRegression(C=0.1).fit(X_train, y_train)
pred_logreg = logreg.predict(X_test)
print("logreg 점수: {:.2f}".format(logreg.score(X_test, y_test)))
```

dummy 점수: 0.80  
logreg 점수: 0.98

무작위로 예측해도 80%를 맞춥니다. 이런 무작위 예측의 성능을 구별할 수 있는 방법이 필요합니다.

# 오차 행렬 confusion matrix

```
In [45]: from sklearn.metrics import confusion_matrix
```

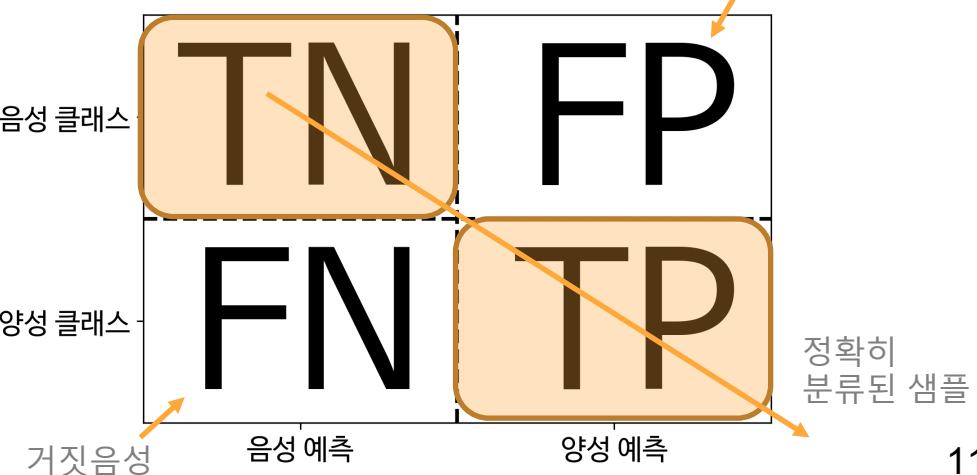
```
confusion = confusion_matrix(y_test, pred_logreg)  
print("오차 행렬:\n{}\n".format(confusion))
```

오차 행렬:

```
[[401  2]  
 [ 8 39]]
```

'9 아님' 정답	401	2
'9' 정답	8	39

'9 아님' 예측      '9' 예측



# 더미 분류기의 오차행렬

```
In [48]: print("빈도 기반 더미 모델:")
print(confusion_matrix(y_test, pred_most_frequent))
print("\n무작위 더미 모델:")
print(confusion_matrix(y_test, pred_dummy))
print("\n결정 트리:")
print(confusion_matrix(y_test, pred_tree))
print("\n로지스틱 회귀")
print(confusion_matrix(y_test, pred_logreg))
```

음성 클래스	TN	FP
양성 클래스	FN	TP
음성 예측		양성 예측

오차행렬을 조사하는 것은  
조금 번거롭고 정성적입니다.

비슷한 FP, FN

가장 좋음

무조건 '9아님'으로 예측(TN,FN)

빈도 기반 더미 모델:  
[[ 403 0 ]  
 [ 47 0 ]]

무작위 더미 모델:  
[[ 360 43 ]  
 [ 40 7 ]]

결정 트리:  
[[ 390 13 ]  
 [ 24 23 ]]

로지스틱 회귀  
[[ 401 2 ]  
 [ 8 39 ]]

비슷한 성능

# 정확도 accuracy, 정밀도 precision, 재현율 recall

$$\text{정확도} = \frac{TP + TN}{TP + TN + FP + FN}$$

TN	FP
FN	TP

양성으로 예측된 것 중  
진짜 양성의 비율(양성 예측도, PPV)  
FP의 수를 줄이는 것이 목표일 때 사용  
ex) 신약의 효과 측정

$$\text{정밀도} = \frac{TP}{TP + FP}$$



TN	FP
FN	TP

진짜 양성 중 양성으로 예측된 비율  
(민감도, 적중률, 진짜양성비율-TPR)  
모든 양성 샘플을 식별해야 할 때 사용  
즉, FN의 수를 줄이는 것이 목표  
ex) 암 진단

$$\text{재현율} = \frac{TP}{TP + FN}$$



TN	FP
FN	TP

# 정밀도 <> 재현율

모두 양성으로 예측하면  
FP가 커져  
정밀도가 낮아지고

FN은 0이 되어  
재현율은 완벽해집니다.

$$\text{정밀도} = \frac{TP}{TP + FP}$$

양성 하나만을 제대로 예측하면  
FP는 0이어서  
정밀도는 1이 되지만

$$\text{재현율} = \frac{TP}{TP + FN}$$

FN이 커져  
재현율은 낮아집니다.

더 다양한 지표는 : [https://en.wikipedia.org/wiki/Precision\\_and\\_recall](https://en.wikipedia.org/wiki/Precision_and_recall)

# f-점수 f-score

정밀도와 재현율의 조화평균으로  $f_1$ -점수라고도 합니다.

불균형 데이터셋에 좋지만 이해하거나 설명하기 어렵습니다.

$$F = 2 \times \frac{\text{정밀도} \times \text{재현율}}{\text{정밀도} + \text{재현율}}$$

f-점수의 분모가 0이 됨

빈도 기반 더미 모델:  
[[403 0]  
 [ 47 0]]

```
In [49]: from sklearn.metrics import f1_score
print("빈도 기반 더미 모델의 f1 score: {:.2f}".format(
    f1_score(y_test, pred_most_frequent)))
print("무작위 더미 모델의 f1 score: {:.2f}".format(f1_score(y_test, pred_dummy)))
print("트리 모델의 f1 score: {:.2f}".format(f1_score(y_test, pred_tree)))
print("로지스틱 회귀 모델의 f1 score: {:.2f}".format(
    f1_score(y_test, pred_logreg)))
```

빈도 기반 더미 모델의 f1 score: 0.00

정확도

무작위 더미 모델의 f1 score: 0.14

0.80

트리 모델의 f1 score: 0.55

0.92

로지스틱 회귀 모델의 f1 score: 0.89

f1 점수로 성능의 차이 확인 가능

# classification\_report + pred\_most\_frequent

정밀도, 재현율, f1 점수를 모두 한번에 출력합니다

```
In [50]: from sklearn.metrics import classification_report  
print(classification_report(y_test, pred_most_frequent,  
                           target_names=[ "9 아님", "9 "]))
```

	precision	recall	f1-score	support
9 아님	0.90	1.00	0.94	403
9	0.00	0.00	0.00	47
avg / total	0.80	0.90	0.85	450

양성 클래스를 '9아님'으로 바꾸었을 때 점수

pred\_most\_frequent는 모든 데이터를 '9아님'으로 예측하기 때문에 양성 클래스를 맞춘 것이 없음

빈도 기반 더미 모델:  
[[ 403 0 ]  
 [ 47 0 ]]

예측값

진짜 타깃값

# classification\_report + pred\_dummy, pred\_logreg

```
In [51]: print(classification_report(y_test, pred_dummy,  
target_names=[ "9 아님", "9" ]))
```

어떤 클래스를  
양성 클래스로  
선택하는지가 중요

	precision	recall	f1-score	support
9 아님	0.90	0.89	0.90	403
9	0.14	0.15	0.14	47
avg / total	0.82	0.82	0.82	450

무작위 더미 모델:  
[[ 360 43]  
[ 40 7]]

```
In [52]: print(classification_report(y_test, pred_logreg,  
target_names=[ "9 아님", "9" ]))
```

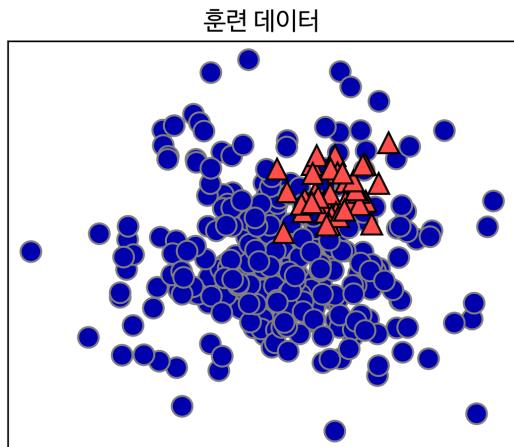
	precision	recall	f1-score	support
9 아님	0.98	1.00	0.99	403
9	0.95	0.83	0.89	47
avg / total	0.98	0.98	0.98	450

로지스틱 회귀  
[[ 401 2]  
[ 8 39]]

# 불확실성

대부분 모델은 예측의 확신을 가늠하기 위해 `decision_function()`의 값이 0보다 크면 양성으로, 또는 `predict_proba()`의 값이 0.5보다 크면 양성으로 판단할 수 있습니다.

이 함수들의 반환값이 임계점에서 멀어질수록 예측에 대한 확신이 높다고 간주할 수 있습니다.

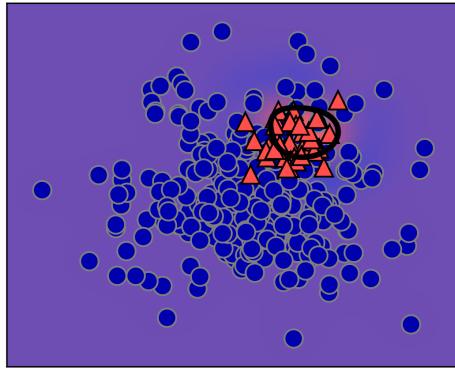


음성 클래스: 400개, 양성 클래스: 50개

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

# decision\_function() > 0

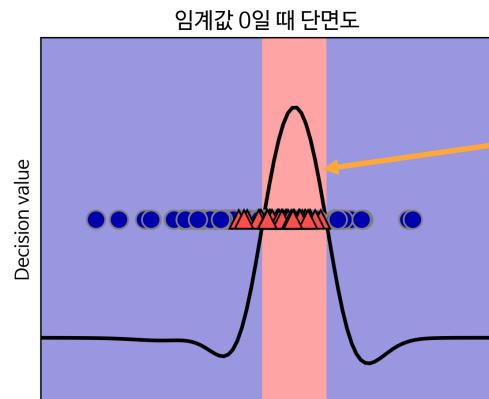
임계값 0일 때



암진단의 예라고 가정하면,

사실 predict() 메서드는 decision\_function() > 0을 사용합니다.

```
In [55]: print(classification_report(y_test, svc.predict(X_test)))
```



임계값 0일 때 단면도

악성

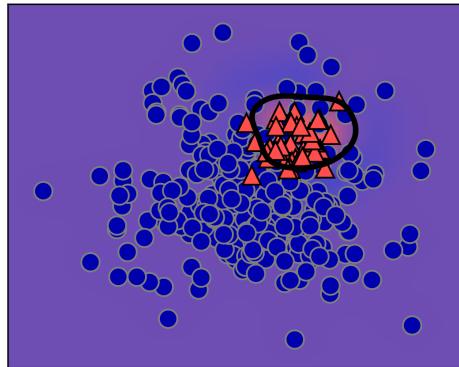
avg / total

	precision	recall	f1-score	support
0	0.97	0.89	0.93	104
1	0.35	0.67	0.46	9
avg / total	0.92	0.88	0.89	113

악성이 양성 클래스일 경우  
모든 악성을 잡아내기 위해 재현율을 높여야 합니다.

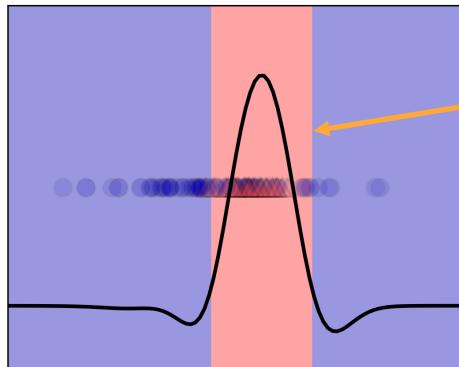
# decision\_function() > -0.8

임계값 -0.8일 때



임계값 -0.8일 때 단면도

Decision value



predict\_proba() 메서드가 있는 모델이라면  
model.predict\_proba(X\_test) > 0.45 이 더 쉬움

0에서 -0.8로 낮추었습니다.

```
In [56]: y_pred_lower_threshold = svc.decision_function(X_test) > -.8
```

```
In [57]: print(classification_report(y_test, y_pred_lower_threshold))
```

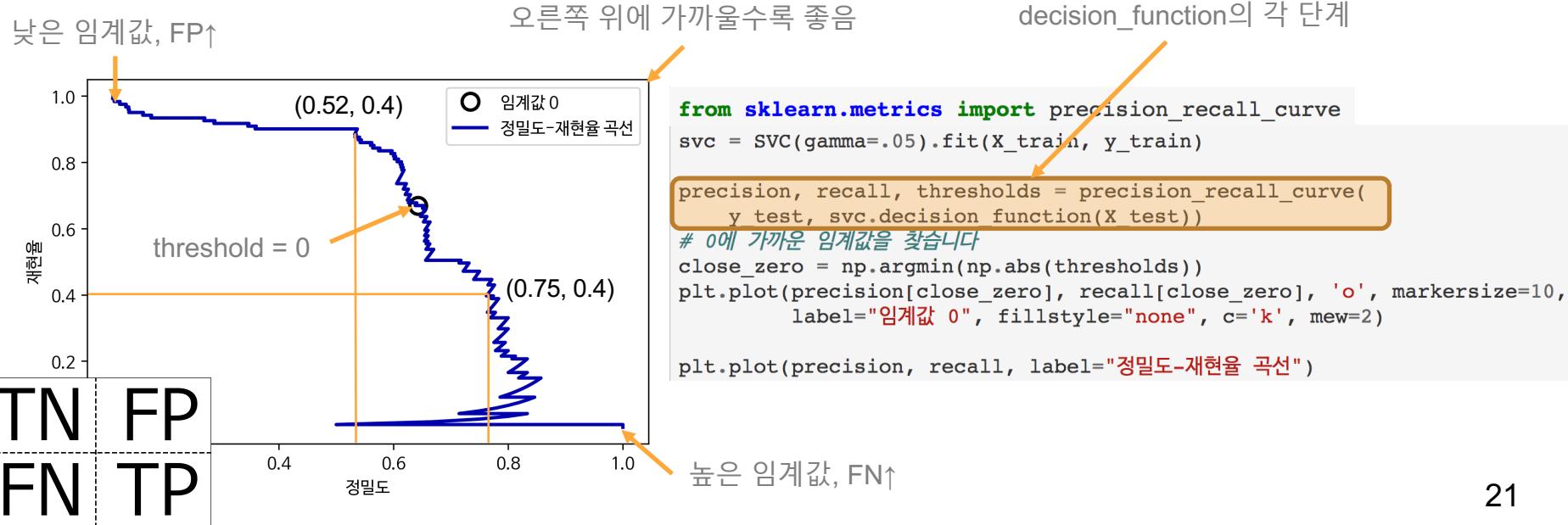
	precision	recall	f1-score	support
0	1.00	0.82	0.90	104
1	0.32	1.00	0.49	9
avg / total	0.95	0.83	0.87	113

실전에서는 테스트 세트를 사용해서는 안됩니다!

# 정밀도-재현율 곡선

임계값을 바꾸는 것은 정밀도와 재현율의 트레이드 오프를 조정하는 일입니다.

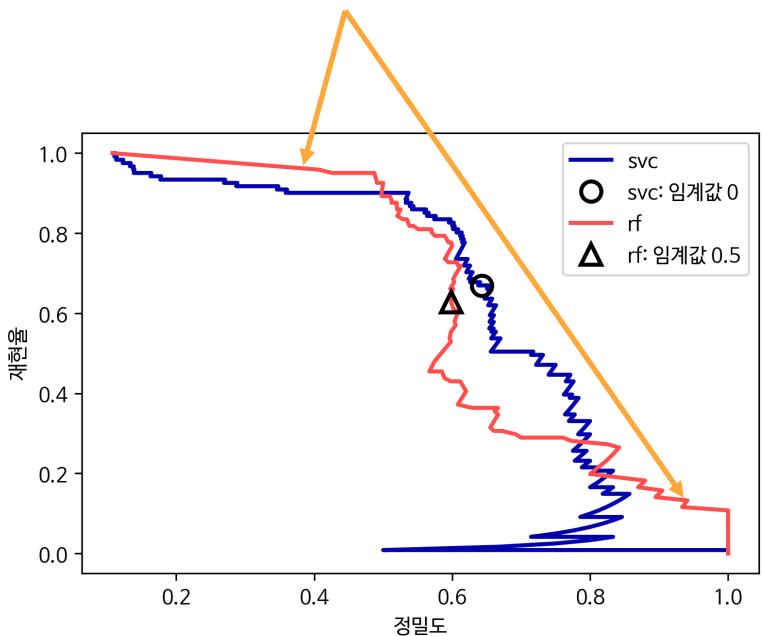
decision\_function과 predict\_proba 함수에서 운영 포인트 operating point 결정할 때 도움이 됩니다.



# RandomForestClassifier vs SVC

In [60]:

재현율이나 정밀도가 극단일 경우  
랜덤 포레스트가 더 낫습니다.



```
from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier(n_estimators=100, random_state=0, max_features=2)
rf.fit(X_train, y_train)

# RandomForestClassifier는 decision function 대신 predict_proba를 제공합니다.
precision_rf, recall_rf, thresholds_rf = precision_recall_curve(
    y_test, rf.predict_proba(X_test)[:, 1])

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)
plt.plot(precision_rf, recall_rf, label="rf")
```

랜덤포레스트는 `decision_function()` 이 없습니다.  
precision\_recall\_curve 함수에 확률값을 전달할 때는  
predict\_proba의 결과중 양성 클래스(배열 인덱스 1)  
에 대한 확률 값을 전달합니다.

# 평균 정밀도 average precision

정밀도-재현율 곡선의 아랫부분 면적을 나타냅니다(0~1 사이).

sklearn.metrics.average\_precision\_score

```
In [61]: print("랜덤 포레스트의 f1_score: {:.3f}".format(
    f1_score(y_test, rf.predict(X_test))))
print("svc의 f1_score: {:.3f}".format(f1_score(y_test, svc.predict(X_test))))
```

랜덤 포레스트의 f1\_score: 0.610  
svc의 f1\_score: 0.656

기본 임계값의 f1 점수를 봐서는 SVC가 더 높습니다.

양성 클래스의 확률값

```
In [62]: from sklearn.metrics import average_precision_score
ap_rf = average_precision_score(y_test, rf.predict_proba(X_test)[:, 1])
ap_svc = average_precision_score(y_test, svc.decision_function(X_test))
print("랜덤 포레스트의 평균 정밀도: {:.3f}".format(ap_rf))
print("svc의 평균 정밀도: {:.3f}".format(ap_svc))
```

랜덤 포레스트의 평균 정밀도: 0.660  
svc의 평균 정밀도: 0.666

결정함수의 반환값

평균 정밀도는 거의 비슷한 수준으로  
f1 점수의 결과와는 조금 다름

# ROC와 AUC

수신기 조작 특성(Receiver operating characteristics)

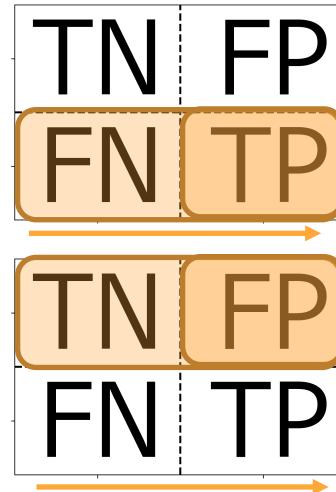
진짜 양성 비율(TPR, 재현율)에 대한 거짓 양성 비율(FPR)의 그래프

`sklearn.metrics.roc_curve`

$$\text{재현율} = \frac{TP}{TP + FN}$$

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

전체 음성 샘플 중에서 거짓  
양성(FP)으로 잘 못 분류된 비율



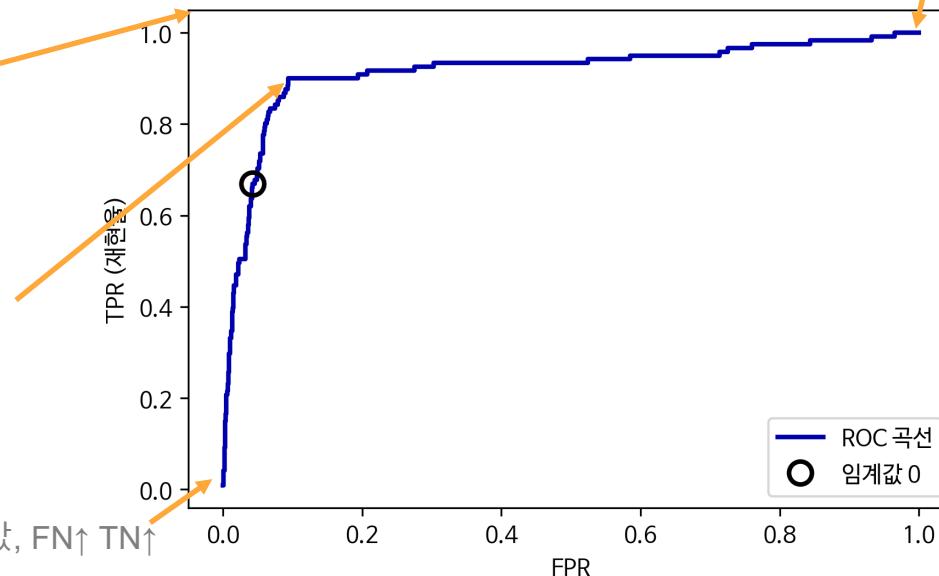
# roc\_curve + SVC

```
In [63]: from sklearn.metrics import roc_curve  
fpr, tpr, thresholds = roc_curve(y_test, svc.decision_function(X_test))  
  
plt.plot(fpr, tpr, label="ROC 곡선")
```

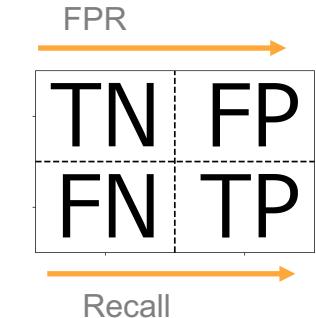
왼쪽 위에 가까울수록 좋음  
거짓 양성(FP)과  
거짓 음성(FN)이 적게

적절한 운영 포인트

높은 임계값, FN↑ TN↑

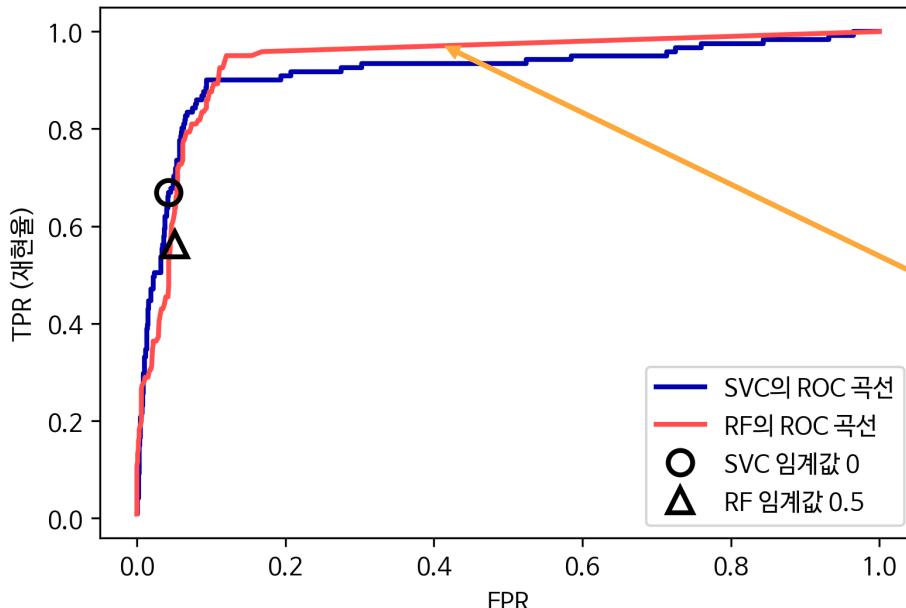


낮은 임계값, FP↑ TP↑



# RandomForestClassifier vs SVC

```
In [64]: 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")
```



랜덤포레스트는 FPR을 조금 더 희생해서 높은 재현율을 얻을 수 있음

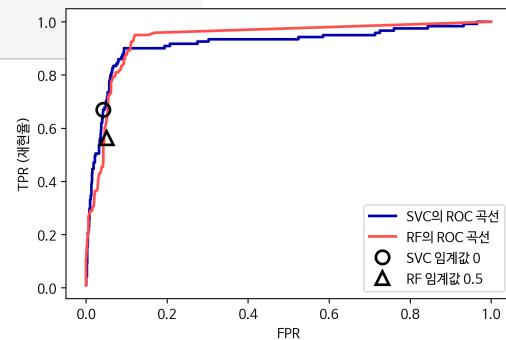
# AUC area under the curve

ROC 곡선 아래의 면적(정밀도-재현율 곡선의 평균 정밀도 처럼)

sklearn.metrics.roc\_auc\_score

```
In [65]: 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: 0.937  
SVC의 AUC: 0.916



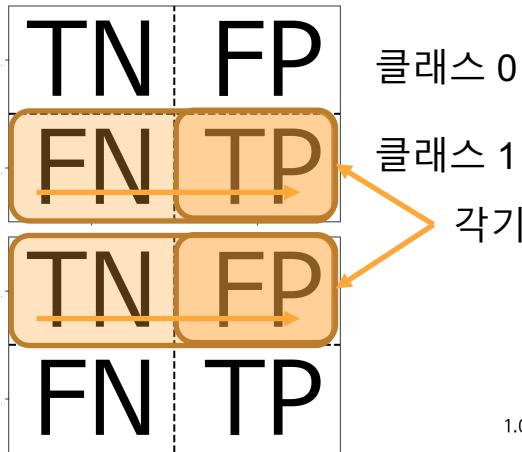
# AUC 특징

불균형한 데이터셋에도  
안정적임

$$\text{재현율} = \frac{TP}{TP + FN}$$

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

불균형 데이터셋에서도 무작위로 선택하면  
TPR, FPR이 비슷해지므로 ROC 곡선이  
y=x 가 되어 AUC는 0.5에 가까워짐

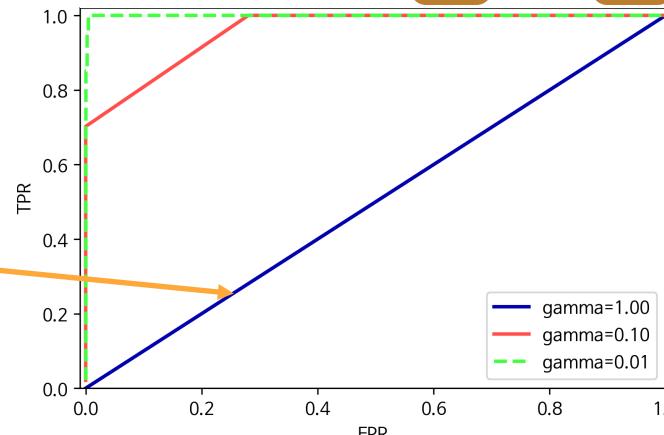


클래스 0

클래스 1

9가 양성인 digits 데이터셋에  
훈련시킨 세개의 SVC 모델

gamma = 1.00	정확도 = 0.90	AUC = 0.50
gamma = 0.10	정확도 = 0.90	AUC = 0.96
gamma = 0.01	정확도 = 0.90	AUC = 1.00



# 다중 분류

# 다중 분류의 평가 지표

이진 분류 평가 지표에서 유도한 것으로 모든 클래스에 대해 평균을 냅니다.

다중 분류에서도 불균형한 데이터셋에서는 정확도가 좋은 지표가 아닙니다.

손글씨 숫자 10개에 대한 오차 행렬

```
from sklearn.metrics import accuracy_score
x_train, x_test, y_train, y_test = train_test_split(
    digits.data, digits.target, random_state=0)
lr = LogisticRegression().fit(x_train, y_train)
pred = lr.predict(x_test)
print("정확도: {:.3f}".format(accuracy_score(y_test, pred)))
print("오차 행렬:\n{}".format(confusion_matrix(y_test, pred)))
```

정확도: 0.953

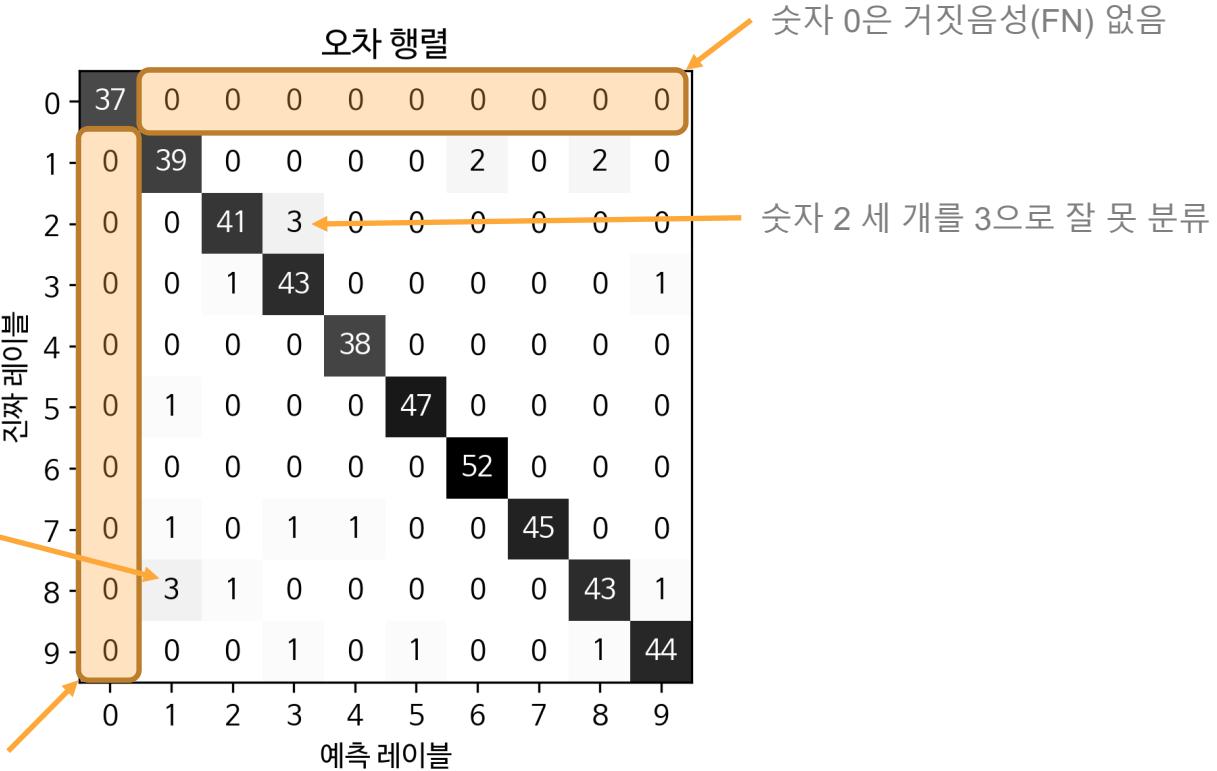
오차 행렬:

37	0	0	0	0	0	0	0	0	0
0	39	0	0	0	0	2	0	2	0
0	0	41	3	0	0	0	0	0	0
0	0	1	43	0	0	0	0	0	1
0	0	0	0	38	0	0	0	0	0
0	1	0	0	0	47	0	0	0	0
0	0	0	0	0	0	52	0	0	0
0	1	0	1	1	0	0	45	0	0
0	3	1	0	0	0	0	0	43	1
0	0	0	1	0	1	0	0	1	44

↑  
타깃값

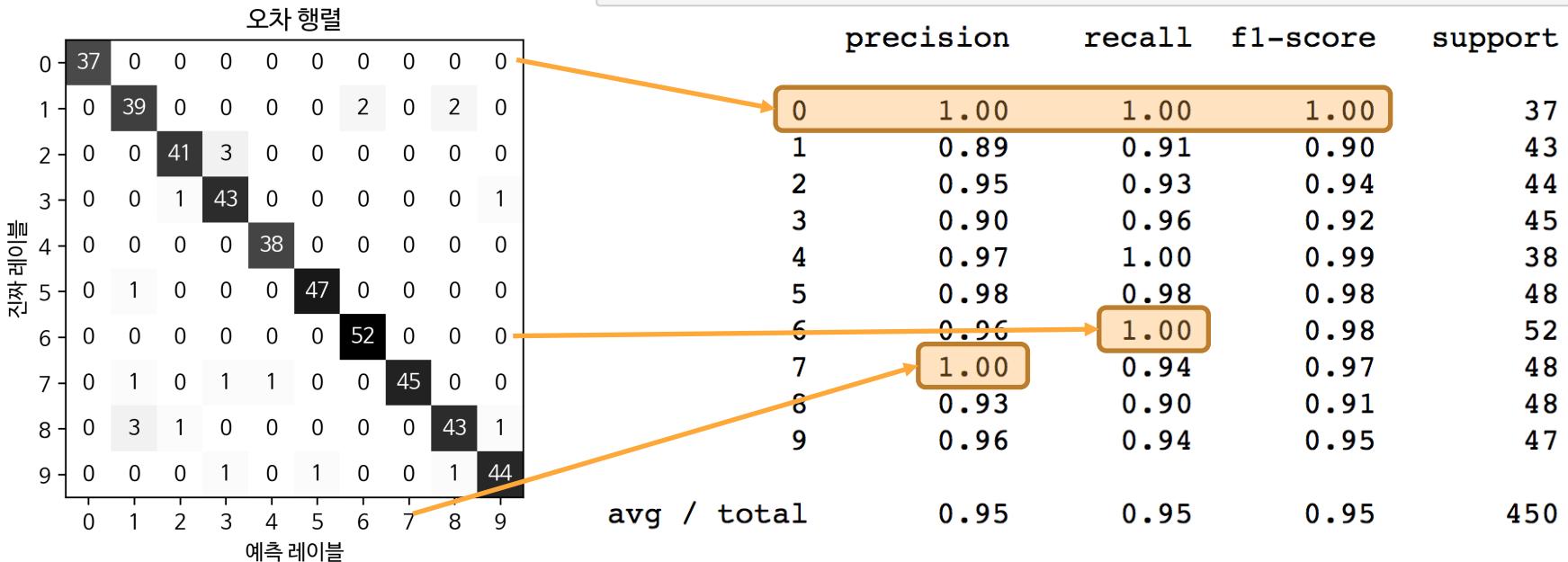
↑  
예측값

# 다중 분류의 오차 행렬



# 다중 분류 리포트

```
In [69]: print(classification_report(y_test, pred))
```



# 다중 분류 $f_1$ -점수

다중 분류용  $f_1$ -점수는 한 클래스를 양성 클래스로 두고 나머지를 음성으로 간주하여 각 클래스마다  $f_1$ -점수를 계산합니다.

```
f1_score(y_test, pred, average="...")
```

- “macro”: 가중치 없이 평균을 냅니다(클래스별 비중을 동일하게 취급).
- “weighted”: 클래스별 샘플 수로 가중치를 두어 평균을 냅니다.(분류 리포트)
- “micro”: 모든 클래스의 FP, FN, TP를 합하여  $f_1$ -점수를 계산합니다(모든 샘플을 똑같이 간주).
- “binary”: 이진 분류에 해당, 기본값.

```
In [70]: print("micro 평균 f1 점수: {:.3f}".format(f1_score(y_test, pred, average="micro")))
      print("macro 평균 f1 점수: {:.3f}".format(f1_score(y_test, pred, average="macro")))
```

```
micro 평균 f1 점수: 0.953
macro 평균 f1 점수: 0.954
```

# 회귀의 평가 지표

$R^2$

회귀는 연속된 값을 예측하므로  $R^2$ 만으로 충분합니다.

$$R^2 = 1 - \frac{\sum(y - \hat{y})^2}{\sum(y - \bar{y})^2}$$

가끔 평균 제곱 예러나 평균 절댓값 에러를 사용하기도 합니다(비즈니스 지표와 연계하기 위해).

$$MSE = \frac{1}{n} \sum (y - \hat{y})^2$$

$$MAE = \frac{1}{n} \sum |y - \hat{y}|^2$$

# 모델 선택과 평가 지표

# cross\_val\_score

scoring 매개변수에 원하는 평가 지표를 지정할 수 있습니다. 기본값은 모델의 score() 메소드입니다.

```
In [71]: # 분류의 기본 평가 지표는 정확도 입니다
print("기본 평가 지표: {}".format(
    cross_val_score(SVC(), digits.data, digits.target == 9)))
# scoring="accuracy"의 결과는 같습니다.
explicit_accuracy = cross_val_score(SVC(), digits.data, digits.target == 9,
                                      scoring="accuracy")
print("정확도 지표: {}".format(explicit_accuracy))
roc_auc = cross_val_score(SVC(), digits.data, digits.target == 9,
                           scoring="roc_auc")
print("AUC 지표: {}".format(roc_auc))
```

기본 평가 지표: [ 0.9 0.9 0.9]

정확도 지표: [ 0.9 0.9 0.9]

AUC 지표: [ 0.994 0.99 0.996]

roc\_auc\_score() 함수를 의미

# GridSearchCV

In [73]: # AUC 지표 사용

```
grid = GridSearchCV(SVC(), param_grid=param_grid, scoring="roc_auc")
grid.fit(X_train, y_train)
print("AUC 지표를 사용한 그리드 서치")
print("최적의 파라미터: ", grid.best_params_)
print("최상의 교차 검증 점수 (AUC): {:.3f}".format(grid.best_score_))
print("테스트 세트 AUC: {:.3f}".format(
    roc_auc_score(y_test, grid.decision_function(X_test))))
print("테스트 세트 정확도: {:.3f}".format(grid.score(X_test, y_test)))
```

AUC 지표를 사용한 그리드 서치

최적의 파라미터: {'gamma': 0.01}

최상의 교차 검증 점수 (AUC): 0.997

테스트 세트 AUC: 1.000

테스트 세트 정확도: 1.000

정확도 지표를 사용한 그리드 서치

최적의 파라미터: {'gamma': 0.0001}

최상의 교차 검증 점수 (정확도): 0.970

테스트 세트 AUC: 0.992

테스트 세트 accuracy: 0.973

# 대표적인 scoring 옵션

분류: accuracy(기본값)  
roc\_auc(ROC 곡선 아래 면적)  
average\_precision(정확도-재현율 곡선 아래 면적)  
f1, f1\_macro, f1\_micro, f1\_weighted

회귀: r2( $R^2$ )  
mean\_square\_error(평균 제곱 오차)  
mean\_absolute\_error(평균 절댓값 오차)

```
In [74]: from sklearn.metrics.scorer import SCORERS
print("가능한 평가 방식:\n{}".format(sorted(SCORERS.keys()))))
```

# 요약 및 정리

# 중요한 주의 사항

교차 검증을 해야 합니다.

훈련 데이터: 모델 학습

검증 데이터: 모델과 매개변수 선택

테스트 데이터: 모델 평가 (마지막에 딱 한번만!)



모델 선택과 평가에 적절한 지표를 사용합니다.

높은 정확도를 가진 모델이 아니라 비즈니스 목표에 맞는 모델이 되어야 합니다.

현실에서는 불균형한 데이터셋이 아주 많습니다.

거짓 양성(FP)과 거짓 음성(FN)이 큰 영향을 미치므로 올바른 평가지표 선택 필요

감사합니다.

-질문-