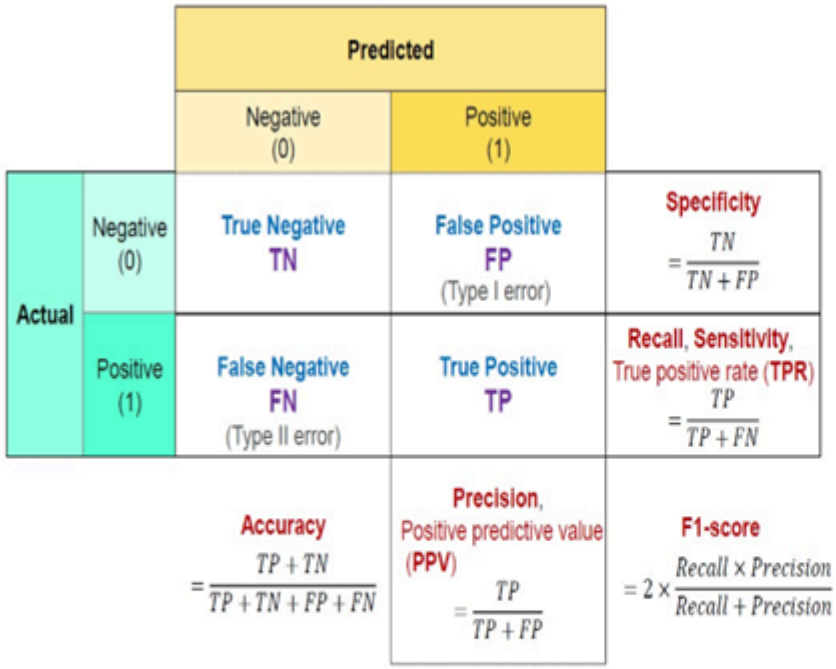
혼동 행렬(Confusion Matrix)과 ROC(Rate Of Change)

Confusion Matrix(혼동 행렬 or 오차 행렬)

- 예측과 실제 값 사이의 관계를 행렬 형태로 표현한 것
- 오차 행렬에서 머신러닝 모델의 성능을 평가하는 여러 지표를 도출할 수 있음



- True : 예측한 것이 정답
- False : 예측한 것이 오답
- Positive : 모델이 Positive라고 예측
  예) 암 진단 시 암이라고 진단
- Negative : 모델이 Negative라고 예측

1. Accuracy (정확도)
2. Precision (정밀도)
3. Recall (재현율)
4. F1-score

- **TP(True Positive)** : 모델이 Positive라고 예측한 것이 정답. 즉, **실제 값은 Positive**
- **FP(False Positive)** : 모델이 Positive라고 예측한 것이 오답. 즉, **실제 값은 Negative → 1종 오류**
- **FN(False Negative)** : 모델이 Negative라고 예측한 것이 오답. 즉, **실제 값은 Positive → 2종 오류**
- **TN(True Negative)** : 모델이 Negative라고 예측한 것이 정답. 즉, **실제 값은 Negative**

---

1) Accuracy(정확도)

- Accuracy = TP + TN / TP + TN + FP + FN
- 전체 문제 중에서 정답을 맞춘 비율(실제 데이터에서 예측 데이터가 얼마나 같은지를 판단하는 지표)

> ◦ 주의 사항 1

        - 이진 분류의 경우 데이터의 구성에 따라 ML(Machine Learning) 모델의 성능을 왜곡할 수 있기 때문에 정확도 수치 하나만 가지고 성능을 평가하지 않음
    - 주의 사항 2
        - 정확도 평가 지표는 불균형한 레이블 데이터 세트에서는 성능 수치로 사용돼서는 안됨

$$\text{정확도(Accuracy)} = \frac{\text{예측 결과가 동일한 데이터 건수}}{\text{전체 예측 데이터 건수}}$$

---

2) Precision(정밀도)

- Precision = TP(True Positive) / (TP(True Positive) + FP(False positive))
- True라고 분류한 것들 중에서 실제로 True인 것의 비율(PPV(Positive Predictive Value) 즉 Positive 정답률이라고도 불림)
- 현실 사례
    - 스팸 메일을 검출할 때 스팸 메일이 아닌데 스팸 메일로 판단해 차단하면 중요한 메일을 못 받을 수 있음
- Precision이 낮다 - True가 아닌데 True라고 한 것이 많음
- Precision이 (과도하게) 높다 - 아주 확실한 경우에만 참으로 예측했음

---

3) Recall(재현율)

- Recall = TP(True Positive) / (TP(True Positive) + FN(False Negative))

- 실제 True인 것 중에서 True라고 예측한 것의 비율(Sensitivity 혹은 Hit rate라고도 불림)
- 현실 사례

> - 지진 지진이 나지 않았지만 지진이라고 예측해 대피명령을 한 것은 생명에 지장이 없지만 지진이 났는데 지진이 나지 않을 것이라고 예측해 대피명령이 없다면 생명이 위험함
> - 암 검출 시 실제로 암에 걸렸는데, 걸리지 않는다고 판단하는 것이 가장 위험함

- Recall이 낮다 - True인데 못 찾은 것이 많음
- Recall이 (과도하게) 높다 - True로 예측한 것이 필요 이상으로 많음(모든 예측을 True로 하면 예측 성능과 상관 없이 Recall이 높게 나올 수 있음)

---

4) F1-score

- Precision과 Recall은 상호보완적이기 때문에, Recall을 올리면 Precision이 내려가고, Precision을 올리면 Recall이 내려갈 수 밖에 없음
- 이를 보완하기 위해 생겨난 것이 Recall과 Precision의 조화평균임

> - 조화평균이란?
>
> > - '역수의 산술평균의 역수'이다.
> > - 역수의 차원에서 평균을 구하고, 다시 역수를 취해 원래 차원의 값으로 돌아오는 것이라고 함

- Precision과 Recall의 조화평균으로 0.0~ 1.0 사이의 값을 가짐
- 값이 1에 가까울수록 좋은 모델임

$$F_1 = \frac{2}{\mathrm{recall}^{-1} + \mathrm{precision}^{-1}} = 2 \cdot \frac{\mathrm{precision} \cdot \mathrm{recall}}{\mathrm{precision} + \mathrm{recall}} = \frac{\mathrm{tp}}{\mathrm{tp} + \frac{1}{2}(\mathrm{fp} + \mathrm{fn})}$$

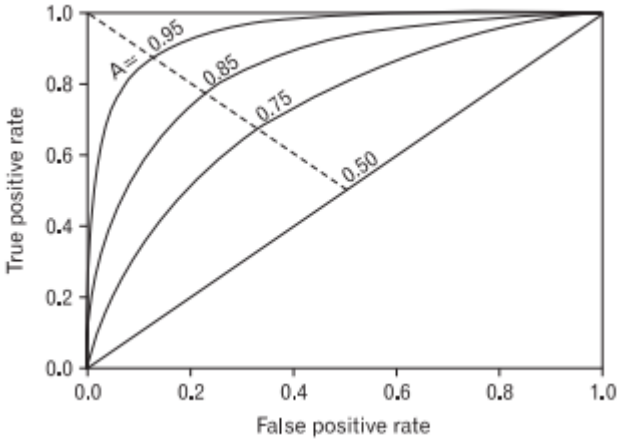
```
#분류 문제에서는 정확도(Accuracy), 정밀도(Precision), 재현율(Recall), F1 점수(F1 Score) 등의 지표를 통해
#모델의 성능을 평가할 수 있음
#이러한 지표들은 모델이 얼마나 잘 예측을 수행하는지에 대한 다양한 측면을 제공
#예를 들어, 정확도는 전체 데이터 중 모델이 올바르게 예측한 비율을 나타내며, 간단하게 모델의 성능을 파악할 수 있는 지표
#하지만, 데이터의 클래스 불균형이 심한 경우에는 정확도만으로는 모델의 성능을 제대로 평가하기 어려움

#정밀도와 재현율은 모델이 얼마나 정확하게 긍정적인 예측을 수행하는지에 대한 정보를 제공

#F1 점수는 정밀도와 재현율의 조화 평균을 나타내며, 두 지표 사이의 균형을 평가하는 데 사용
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

# 예측 결과와 실제 레이블
y_pred = [0, 2, 1, 3]
y_true = [0, 1, 2, 3]

# 성능 지표 계산
accuracy = accuracy_score(y_true, y_pred)
precision = precision_score(y_true, y_pred, average='macro')
recall = recall_score(y_true, y_pred, average='macro')
f1 = f1_score(y_true, y_pred, average='macro')

print(f'Accuracy: {accuracy}WnPrecision: {precision}WnRecall: {recall}WnF1 Score: {f1}')
```

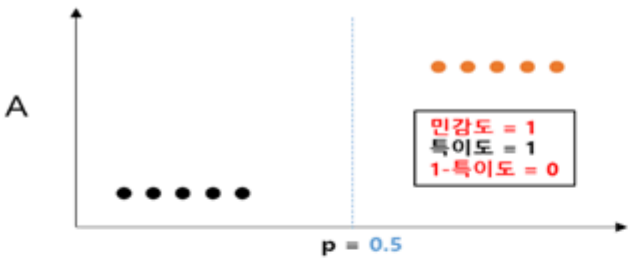
```
Accuracy: 0.5
Precision: 0.5
Recall: 0.5
F1 Score: 0.5
```

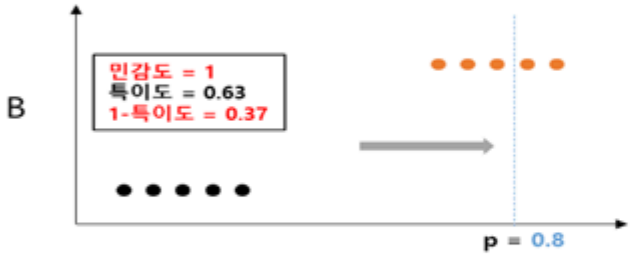
ROC(Receiver Operating Characteristic)

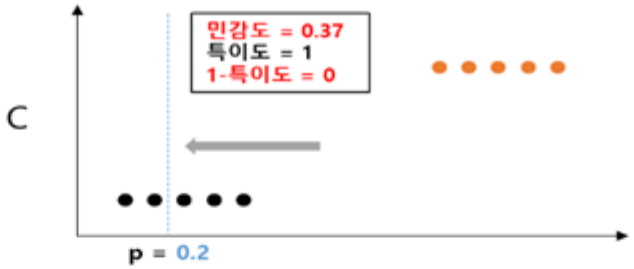
- 분류 예측 모형의 성능을 평가할 때 사용

  - ROC Curve는 모든 분류 임계값에서 분류 모델의 성능을 보여주는 그래프로 x축이 FPR(1-특이도), y축이 TPR(민감도)인 그래프
  - ROC Curve는 AUC(Area Under Curve : 그래프 아래의 면적)를 이용해 모델의 성능을 평가
  - AUC가 클수록 정확히 분류함을 뜻함
    - 다음 그래프에서 A = 0.95인 모델은 95%의 확률로 제대로 분류하고 있음을 뜻함

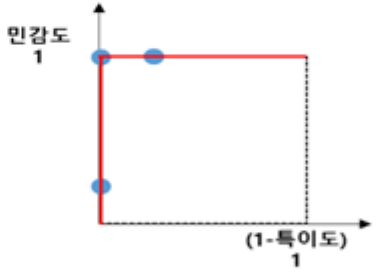








[그림 2] 좋은 모델에서 임계값에 따른 민감도와 특이도 변화



아래 <그림 2>는 아주 좋은 모델인 경우이다. A 차트는 실제 데이터의 분포를 나타내는데 주황색(감염자 : 관심범주)과 검은색(비감염자)이 모델이 예측하는 확률 0.5를 기준으로 분명하게 나뉘어져 있다. 예측은 확률을 기준으로 판정하므로 0.5이상에 해당되는 주황색(실제 감염자)을 모두 감염자라고 판정할 것이고, 그 결과 민감도 =1, 확률 0.5 이하인 검은색은 모두 비감염자라고 예측할 것이므로 특이도 = 1 이다.

반면 B, C는 예측 임계값을 늘이거나 줄인경우인데 예를 들어 B의 경우 임계값을 0.8로 잡으면 0.8이하인 8개의 데이터(검은색 5개, 주황색3)를 모두 비감염자라고 예측할 것이다. 그러나 실제로 비감염자는 검은색 5개 이므로 특이도는 5/8 = 0.63이 된다. C는 반대의 경우다.

이렇게 A, B, C처럼 임계값을 변화시켜 민감도 VS (1-특이도) 쌍을 차트로 그린 것이 ROC곡선이다. <그림 3>을 보면 민감도 VS (1-특이도) 값이 빨간색 축에 붙어서 움직이는 것을 알 수 있다. 따라서 좋은 모델이 만들어 내는 ROC 곡선은 <그림 1>의 파란색 곡선처럼 만들어진다.

AUC(Area Under Curve : 그래프 아래의 면적)

- 분류 예측 모형의 성능을 평가할 때 사용
- ROC커브와 직선 사이의 면적을 의미
- AUC 값의 범위는 0~1이며 값이 클수록 예측의 정확도가 높다고 할 수 있음

```
#Python scikit-learn ROC curve, AUC
#데이터셋을 데이터프레임 형태로 불러옴
import pandas as pd
from sklearn import datasets

# 유방암 데이터셋 로드
data = datasets.load_breast_cancer()
df = pd.DataFrame(data.data, columns = data.feature_names)

# 1이면 양성 종양, 0이면 악성 종양 라벨 컬럼 지정
df['target'] = data.target
df
```

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst texture	w perim
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.30010	0.14710	0.2419	0.07871	...	17.33	1
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.08690	0.07017	0.1812	0.05667	...	23.41	1
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.19740	0.12790	0.2069	0.05999	...	25.53	1
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.24140	0.10520	0.2597	0.09744	...	26.50	
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.19800	0.10430	0.1809	0.05883	...	16.67	1
...	...	...	...	...	...	...	...	...	...	...	...	...	
564	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	0.13890	0.1726	0.05623	...	26.40	1
565	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	0.09791	0.1752	0.05533	...	38.25	1
566	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	0.05302	0.1590	0.05648	...	34.12	1
567	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140	0.15200	0.2397	0.07016	...	39.42	1
568	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000	0.00000	0.1587	0.05884	...	30.37	

569 rows × 31 columns

```
#불러온 데이터셋에 대하여 학습용 데이터와 테스트용 데이터를 분리
#여기서는 target 컬럼을 제외한 모든 열들을 feature로 사용하는 상황을 가정
from sklearn.model_selection import train_test_split

# train, test 셋 분리
X = df.iloc[:, :-1] # target column을 제외한 모든 column을 feature로 사용
y = df['target']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 42)
```

```
#예시 분류 모델로 선형 회귀 모델을 고르고 분류기를 학습
from sklearn.linear_model import LinearRegression

# 선형 회귀 모델 학습
model = LinearRegression()
model.fit(X_train, y_train)
```

```
▾ LinearRegression ⓘ ?
LinearRegression()
```

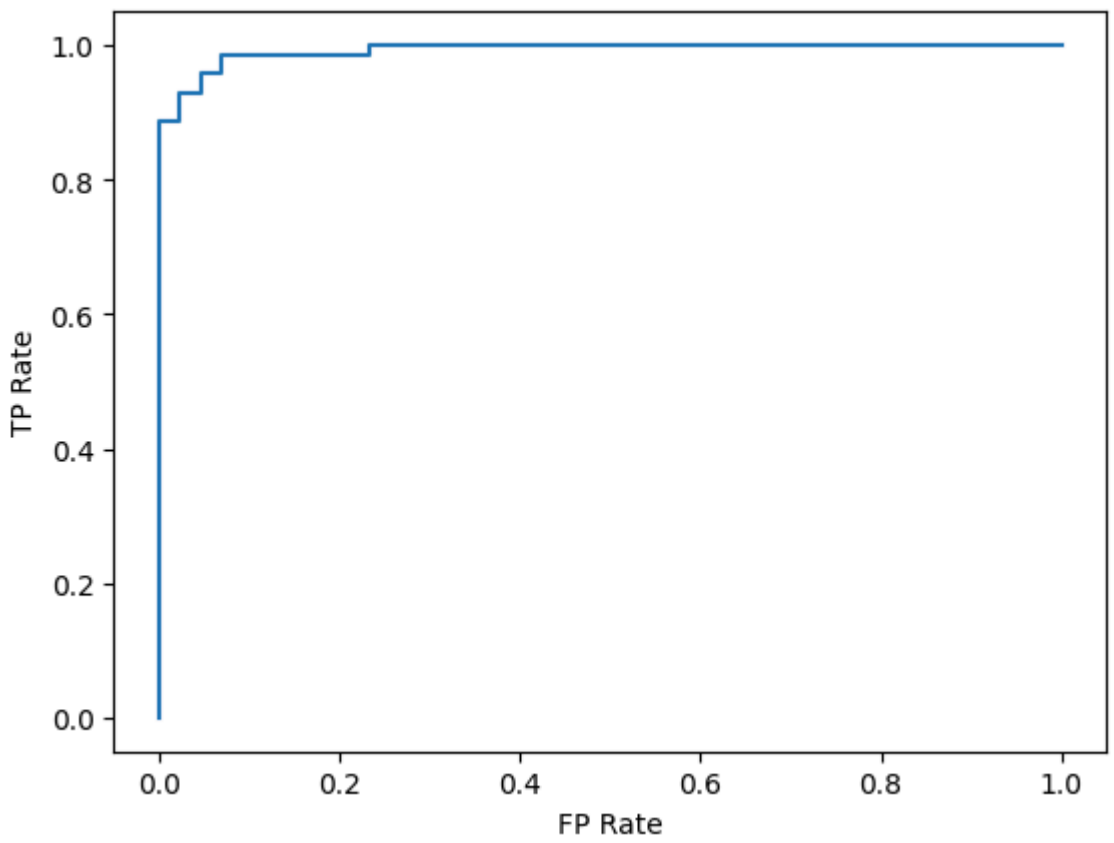
```
#테스트셋 예측 결과에 대한 ROC 커브를 그리기 위하여
#테스트셋 데이터에 대한 라벨 예측 값을 받아옴
y_score = model.predict(X_test) # 테스트 데이터셋에 대한 예측 값

# 앞 5개 값 출력 예시
print(y_score[:5]) # [0.62168355 0.17803853 0.27512068 1.1320503  1.11698318]
```

```
[0.62168355 0.17803853 0.27512068 1.1320503  1.11698318]
```

```
#사이킷런의 roc_curve 함수로 예상 threshold별 FP rate 및 TP rate를 바로 구할 수 있음
#input은 (실제 라벨, 예측 값) 순서로 지정해
#출력 값은 순서대로 FP rate, TP rate, threshold value를 의미
#예측 회귀 값이 threshold보다 크면 1, 아니면 0으로 분류하는 원리
from sklearn.metrics import roc_curve

fpr, tpr, thresholds = roc_curve(y_test, y_score) # input 순서 : 실제 라벨, 예측 값
```

```
#matplotlib 라이브러리를 이용하여 ROC curve를 그림
import matplotlib.pyplot as plt

plt.plot(fpr, tpr)

plt.xlabel('FP Rate')
plt.ylabel('TP Rate')
```

```
8
9 plt.show()
```



```
#y = x 그래프와 ROC 커브를 통해 비교가 용이
plt.plot(fpr, tpr, color = 'red', label = 'ROC curve')
plt.plot([0, 1], [0, 1], color = 'blue', label = 'y = x') # y = x 직선 표시

plt.xlabel('FP Rate')
plt.ylabel('TP Rate')

plt.legend() # 그래프 라벨 표시

plt.show()
```