분류 모델 평가지표 (Classification Model Evaluation Metrics)

혼동 행렬 (Confusion Matrix)

- 분류 모델의 성능을 평가하는 데 사용되는 표
- 실제 클래스와 모델이 예측한 클래스 간의 관계를 시각화
- 이진 분류 문제에서 주로 사용되지만, 다중 클래스 분류 문제에도 확장하여 적용 가능
- 4가지 주요 값으로 구성:
    - **True Positive (TP)**: 실제 True인 것을 모델이 True로 올바르게 예측한 경우
    - **True Negative (TN)**: 실제 False인 것을 모델이 False로 올바르게 예측한 경우
    - **False Positive (FP)**: 실제 False인 것을 모델이 True로 잘못 예측한 경우 (1종 오류, Type I Error)
    - **False Negative (FN)**: 실제 True인 것을 모델이 False로 잘못 예측한 경우 (2종 오류, Type II Error)

주의사항

- 혼동 행렬의 레이블 순서(양성/음성 클래스)를 명확히 정의해야 합니다. 일반적으로 긍정 클래스를 1(또는 True)로, 부정 클래스를 0(또는 False)으로 설정합니다.
- 다중 클래스 분류의 경우, 각 클래스에 대한 이진 분류 문제로 확장하여 해석하거나, 전체 행렬을 통해 각 클래스별 성능을 분석할 수 있습니다.

```
import numpy as np
from sklearn.metrics import confusion_matrix
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# 예시 데이터
y_true = np.array([0, 1, 0, 1, 0, 1, 0, 0, 1, 1])
y_pred = np.array([0, 1, 1, 1, 0, 0, 0, 1, 1, 1])

# scikit-learn을 이용한 혼동 행렬
cm = confusion_matrix(y_true, y_pred)
print("혼동 행렬 (scikit-learn):", cm)

# 혼동 행렬 시각화 (seaborn)
plt.figure(figsize=(6, 4))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", cbar=False,
            xticklabels=['Predicted Negative', 'Predicted Positive'],
            yticklabels=['Actual Negative', 'Actual Positive'])
plt.xlabel('예측 클래스')
plt.ylabel('실제 클래스')
plt.title('혼동 행렬')
plt.show()

# 다중 클래스 분류 예시
y_true_multi = np.array([0, 1, 2, 0, 1, 2, 0, 1, 2])
y_pred_multi = np.array([0, 1, 1, 0, 2, 2, 0, 0, 2])
```

```
cm_multi = confusion_matrix(y_true_multi, y_pred_multi)
print("다중 클래스 혼동 행렬:", cm_multi)

plt.figure(figsize=(7, 5))
sns.heatmap(cm_multi, annot=True, fmt="d", cmap="Greens", cbar=False,
            xticklabels=['Class 0', 'Class 1', 'Class 2'],
            yticklabels=['Class 0', 'Class 1', 'Class 2'])
plt.xlabel('예측 클래스')
plt.ylabel('실제 클래스')
plt.title('다중 클래스 혼동 행렬')
plt.show()
```

결과 해석 방법

- 혼동 행렬의 각 셀은 다음과 같이 해석됩니다:
  - `cm[0, 0]` (좌측 상단): TN (실제 0, 예측 0)
  - `cm[0, 1]` (우측 상단): FP (실제 0, 예측 1)
  - `cm[1, 0]` (좌측 하단): FN (실제 1, 예측 0)
  - `cm[1, 1]` (우측 하단): TP (실제 1, 예측 1)
- 시각화된 혼동 행렬에서 대각선 요소(TP, TN)가 높을수록 모델의 예측 정확도가 높다고 볼 수 있습니다.
- 대각선이 아닌 요소(FP, FN)는 모델의 오류를 나타내며, 어떤 유형의 오류가 더 많은지 파악할 수 있습니다. 예를 들어, FP가 높으면 모델이 실제 음성인 것을 양성으로 잘못 판단하는 경우가 많다는 의미이고, FN이 높으면 실제 양성인 것을 음성으로 잘못 판단하는 경우가 많다는 의미입니다.

장단점 및 대안

- **장점**:
  - 분류 모델의 성능을 직관적으로 이해할 수 있습니다.
  - 모델이 어떤 유형의 오류를 범하는지 상세하게 파악할 수 있어, 모델 개선 방향을 설정하는 데 도움이 됩니다.
  - 불균형 데이터셋에서도 각 클래스별 성능을 개별적으로 평가할 수 있는 기반을 제공합니다.
- **단점**:
  - 다중 클래스 분류 문제에서는 행렬의 크기가 커져 해석이 복잡해질 수 있습니다.
  - 혼동 행렬 자체로는 단일 성능 지표를 제공하지 않으므로, 이를 기반으로 Accuracy, Precision, Recall, F1-Score 등을 추가로 계산해야 합니다.
- **대안**:
  - 혼동 행렬을 기반으로 계산되는 Accuracy, Precision, Recall, F1-Score, ROC AUC 등의 지표를 함께 사용하여 모델 성능을 종합적으로 평가합니다.
  - 다중 클래스 분류의 경우, `classification_report` 함수를 사용하여 각 클래스별 Precision, Recall, F1-Score를 한눈에 확인할 수 있습니다.

Accuracy (정확도)

- 전체 예측 중에서 올바르게 예측한 비율
- 가장 직관적인 분류 모델 평가지표
- $(TP + TN) / (TP + TN + FP + FN)$

주의사항

- 불균형 데이터셋에서는 정확도가 높은 것처럼 보일 수 있지만,
  실제로는 소수 클래스에 대한 예측 성능이 매우 낮을 수 있으므로 주의 필요
- 예를 들어, 99%가 음성 클래스인 데이터셋에서 모델이 모든 것을 음성으로 예측하면 정확도는 99%가 되지만, 실제 양성 클래스는 전혀 맞추지 못하게 됩니다.

```
import numpy as np
from sklearn.metrics import accuracy_score, confusion_matrix

# 예시 데이터
y_true = np.array([0, 1, 0, 1, 0, 1, 0, 0, 1, 1])
y_pred = np.array([0, 1, 1, 1, 0, 0, 0, 1, 1, 1])

# scikit-learn을 이용한 정확도 계산
accuracy = accuracy_score(y_true, y_pred)
print(f"정확도 (Accuracy): {accuracy:.4f}")          # 0.7000

# 혼동 행렬을 이용한 수동 계산
cm = confusion_matrix(y_true, y_pred)
TP = cm[1, 1]
TN = cm[0, 0]
FP = cm[0, 1]
FN = cm[1, 0]
manual_accuracy = (TP + TN) / (TP + TN + FP + FN)
print(f"수동 계산 정확도: {manual_accuracy:.4f}")     # 0.7000
```

결과 해석 방법

- 정확도 값이 1에 가까울수록 모델의 성능이 좋다고 해석할 수 있습니다.
- 0.5보다 높으면 무작위 예측보다 성능이 좋다고 볼 수 있습니다.
- 불균형 데이터셋에서는 다른 평가지표(Precision, Recall, F1-Score 등)와 함께 해석해야 합니다.

장단점 및 대안

- **장점**:
  - 이해하기 쉽고 직관적입니다.
  - 계산이 간단합니다.
- **단점**:
  - 불균형 데이터셋에서 모델의 실제 성능을 왜곡할 수 있습니다.
  - 특정 클래스에 대한 모델의 성능을 파악하기 어렵습니다.
- **대안**:
  - 불균형 데이터셋에서는 Precision, Recall, F1-Score, ROC AUC 등 다른 평가지표를 함께 고려해야 합니다.
  - 다중 클래스 분류에서는 `balanced_accuracy_score`를 사용하여 클래스 불균형의 영향을 완화할 수 있습니다.

Precision (정밀도, 신뢰도)

- 모델이 '양성'이라고 예측한 것 중에서 실제로 '양성'인 비율

- FP(False Positive) 오류를 줄이는 것이 중요한 상황에서 유용
- $TP / (TP + FP)$

적용 가능한 상황

- 오탐(False Positive)이 심각한 결과를 초래하는 경우 (예: 스팸 메일 분류에서 정상 메일을 스팸으로 분류하는 경우, 암 진단에서 건강한 사람을 암 환자로 오진하는 경우).
- 모델이 '양성'으로 예측한 결과에 대한 신뢰도가 중요할 때 사용됩니다.

주의사항

- 정밀도가 높다는 것은 모델이 양성으로 예측한 결과는 대부분 실제 양성이라는 의미
  but, 실제 양성을 얼마나 잘 찾아냈는지(Recall)에 대한 정보는 제공하지 않음
- 클래스 불균형이 심한 경우, 소수 클래스에 대한 정밀도를 평가할 때 특히 유의

함수 인자의 종류 설명 (precision_score)

- `y_true`: 실제 레이블 (정답).
- `y_pred`: 모델의 예측 레이블.
- `labels`: (선택 사항) 계산에 포함할 레이블 목록.
- `pos_label`: (선택 사항) 이진 분류에서 양성 클래스로 간주할 레이블. 기본값은 1입니다.
- `average`: (선택 사항) 다중 클래스/다중 레이블 타겟의 경우, 평균화 방법을 지정합니다.
  - `'binary'`: 이진 분류에만 해당. `pos_label`로 지정된 클래스의 정밀도를 반환합니다.
  - `'micro'`: 전체 TP, FP를 합산하여 정밀도를 계산합니다. (전체 정확도와 동일)
  - `'macro'`: 각 레이블의 정밀도를 계산한 다음 평균을 취합니다. 레이블 불균형을 고려하지 않습니다.
  - `'weighted'`: 각 레이블의 정밀도를 계산한 다음, 각 레이블의 실제 샘플 수에 따라 가중 평균을 취합니다. 레이블 불균형을 고려합니다.
  - `'samples'`: 다중 레이블 분류에만 해당. 각 샘플의 정밀도를 계산한 다음 평균을 취합니다.
  - `None`: 각 클래스별 정밀도 점수의 배열을 반환합니다.
- `sample_weight`: (선택 사항) 샘플 가중치.
- `zero_division`: (선택 사항) 분모가 0일 때 반환할 값. 기본값은 'warn'이며, 0으로 나눌 때 경고를 발생시키고 0을 반환합니다.

```
import numpy as np
from sklearn.metrics import precision_score, confusion_matrix

# 예시 데이터
y_true = np.array([0, 1, 0, 1, 0, 1, 0, 0, 1, 1])
y_pred = np.array([0, 1, 1, 1, 0, 0, 0, 1, 1, 1])

# scikit-learn을 이용한 정밀도 계산
precision = precision_score(y_true, y_pred)
print(f"정밀도 (Precision): {precision:.4f}")        # 0.6667

# 혼동 행렬을 이용한 수동 계산
cm = confusion_matrix(y_true, y_pred)
TP = cm[1, 1]
FP = cm[0, 1]
```

```
manual_precision = TP / (TP + FP)
print(f"수동 계산 정밀도: {manual_precision:.4f}")    # 0.6667

# 다중 클래스 분류에서의 정밀도 (average 파라미터)
y_true_multi = np.array([0, 1, 2, 0, 1, 2, 0, 1, 2])
y_pred_multi = np.array([0, 1, 1, 0, 2, 2, 0, 0, 2])

# 'macro': 각 클래스별 정밀도를 계산한 후 평균 (클래스 불균형에 민감)
precision_macro = precision_score(y_true_multi, y_pred_multi, average='macro')
print(f"다중 클래스 정밀도 (macro): {precision_macro:.4f}")          # 0.6389

# 'weighted': 각 클래스별 정밀도를 계산한 후 샘플 수에 따라 가중 평균
precision_weighted = precision_score(y_true_multi, y_pred_multi,
average='weighted')
print(f"다중 클래스 정밀도 (weighted): {precision_weighted:.4f}")    # 0.6389

# 'micro': 전체 TP, FP, FN을 합산하여 계산 (Accuracy와 동일)
precision_micro = precision_score(y_true_multi, y_pred_multi, average='micro')
print(f"다중 클래스 정밀도 (micro): {precision_micro:.4f}")          # 0.6667

# 'None': 각 클래스별 정밀도 반환
precision_per_class = precision_score(y_true_multi, y_pred_multi, average=None)
print(f"다중 클래스 정밀도 (클래스별): {precision_per_class}")        # [0.75
0.5        0.66666667]
```

결과 해석 방법

- 정밀도 값이 1에 가까울수록 모델이 양성으로 예측한 결과의 신뢰도가 높다고 해석할 수 있습니다.
- 예를 들어, 정밀도가 0.8이라면 모델이 양성으로 예측한 100개 중 80개가 실제로 양성이라는 의미입니다.
- 오탐(FP) 비용이 높은 시나리오에서 중요한 지표입니다.

장단점 및 대안

- **장점**:
  - 모델이 양성으로 예측한 결과의 '정확성'을 평가하는 데 유용합니다.
  - 오탐 비용이 높은 상황에서 모델 선택의 중요한 기준이 됩니다.
- **단점**:
  - 실제 양성인 것들을 얼마나 놓쳤는지(FN)에 대한 정보는 제공하지 않습니다. 즉, 재현율(Recall)이 낮을 수 있습니다.
  - 불균형 데이터셋에서 소수 클래스의 정밀도가 낮게 나올 수 있습니다.
- **대안**:
  - 재현율(Recall)과 함께 고려하여 모델의 전반적인 성능을 평가해야 합니다.
  - Precision-Recall Curve를 통해 다양한 임계값에서의 정밀도와 재현율의 관계를 시각적으로 분석할 수 있습니다.

Recall (재현율, 민감도)

- 실제 '양성'인 것 중에서 모델이 '양성'으로 올바르게 예측한 비율을 나타냅니다.
- FN(False Negative) 오류를 줄이는 것이 중요한 상황에서 유용합니다.
- $TP / (TP + FN)$

적용 가능한 상황

- 미탐(False Negative)이 심각한 결과를 초래하는 경우 (예: 암 진단에서 실제 암 환자를 건강하다고 오진하는 경우, 금융 사기 탐지에서 실제 사기를 정상 거래로 판단하는 경우).
- 실제 양성인 샘플을 놓치지 않는 것이 중요할 때 사용됩니다.

주의사항

- 재현율이 높다는 것은 모델이 실제 양성인 것들을 잘 찾아낸다는 의미
  but, 음성인 것을 양성으로 잘못 판단하는 경우(FP)가 많을 수 있음
  즉, 정밀도(Precision)가 낮을 수 있음
- 클래스 불균형이 심한 경우, 소수 클래스에 대한 재현율을 평가할 때 특히 유의

함수 인자의 종류 설명 (recall_score)

- `y_true`: 실제 레이블 (정답).
- `y_pred`: 모델의 예측 레이블.
- `labels`: (선택 사항) 계산에 포함할 레이블 목록.
- `pos_label`: (선택 사항) 이진 분류에서 양성 클래스로 간주할 레이블. 기본값은 1입니다.
- `average`: (선택 사항) 다중 클래스/다중 레이블 타겟의 경우, 평균화 방법을 지정합니다. `precision_score`와 동일합니다.
- `sample_weight`: (선택 사항) 샘플 가중치.
- `zero_division`: (선택 사항) 분모가 0일 때 반환할 값. 기본값은 'warn'이며, 0으로 나눌 때 경고를 발생시키고 0을 반환합니다.

```
import numpy as np
from sklearn.metrics import recall_score, confusion_matrix

# 예시 데이터
y_true = np.array([0, 1, 0, 1, 0, 1, 0, 0, 1, 1])
y_pred = np.array([0, 1, 1, 1, 0, 0, 0, 1, 1, 1])

# scikit-learn을 이용한 재현율 계산
recall = recall_score(y_true, y_pred)
print(f"재현율 (Recall): {recall:.4f}")          # 0.8000

# 혼동 행렬을 이용한 수동 계산
cm = confusion_matrix(y_true, y_pred)
TP = cm[1, 1]
FN = cm[1, 0]
manual_recall = TP / (TP + FN)
print(f"수동 계산 재현율: {manual_recall:.4f}")  # 0.8000

# 다중 클래스 분류에서의 재현율 (average 파라미터)
y_true_multi = np.array([0, 1, 2, 0, 1, 2, 0, 1, 2])
y_pred_multi = np.array([0, 1, 1, 0, 2, 2, 0, 0, 2])

# 'macro': 각 클래스별 재현율을 계산한 후 평균 (클래스 불균형에 민감)
recall_macro = recall_score(y_true_multi, y_pred_multi, average='macro')
print(f"다중 클래스 재현율 (macro): {recall_macro:.4f}")      # 0.6667
```

```
# 'weighted': 각 클래스별 재현율을 계산한 후 샘플 수에 따라 가중 평균
recall_weighted = recall_score(y_true_multi, y_pred_multi, average='weighted')
print(f"다중 클래스 재현율 (weighted): {recall_weighted:.4f}")    # 0.6667

# 'micro': 전체 TP, FN을 합산하여 계산 (Accuracy와 동일)
recall_micro = recall_score(y_true_multi, y_pred_multi, average='micro')
print(f"다중 클래스 재현율 (micro): {recall_micro:.4f}")      # 0.6667

# 'None': 각 클래스별 재현율 반환
recall_per_class = recall_score(y_true_multi, y_pred_multi, average=None)
print(f"다중 클래스 재현율 (클래스별): {recall_per_class}")       # [1.
0.33333333 0.66666667]
```

결과 해석 방법

- 재현율 값이 1에 가까울수록 모델이 실제 양성인 샘플들을 놓치지 않고 잘 찾아낸다고 해석할 수 있습니다.
- 예를 들어, 재현율이 0.9라면 실제 양성인 100개 중 90개를 모델이 양성으로 올바르게 예측했다는 의미입니다.
- 미탐(FN) 비용이 높은 시나리오에서 중요한 지표입니다.

장단점 및 대안

- **장점**:
  - 모델이 실제 양성 클래스를 얼마나 잘 '감지'하는지 평가하는 데 유용합니다.
  - 미탐 비용이 높은 상황에서 모델 선택의 중요한 기준이 됩니다.
- **단점**:
  - 모델이 음성인 것을 양성으로 잘못 판단하는 경우(FP)가 많아도 재현율은 높게 나올 수 있습니다. 즉, 정밀도(Precision)가 낮을 수 있습니다.
  - 불균형 데이터셋에서 소수 클래스의 재현율이 낮게 나올 수 있습니다.
- **대안**:
  - 정밀도(Precision)와 함께 고려하여 모델의 전반적인 성능을 평가해야 합니다.
  - Precision-Recall Curve를 통해 다양한 임계값에서의 정밀도와 재현율의 관계를 시각적으로 분석할 수 있습니다.

F1-Score (F1 점수)

- 정밀도(Precision)와 재현율(Recall)의 조화 평균
- 정밀도와 재현율이 모두 중요한 상황에서 두 지표의 균형을 평가하는 데 사용
- $2 * (Precision * Recall) / (Precision + Recall)$

적용 가능한 상황

- 정밀도와 재현율 중 어느 한쪽으로 치우치지 않고, 두 지표 모두 높은 성능을 요구하는 경우 (예: 검색 엔진의 검색 결과, 의료 진단 시스템).
- 특히 클래스 불균형이 있는 데이터셋에서 모델의 성능을 평가할 때 정확도보다 더 신뢰할 수 있는 지표로 사용됩니다.

주의사항

- F1-Score는 정밀도와 재현율이 모두 높을 때 높은 값을 가집니다. 둘 중 하나라도 낮으면 F1-Score도 낮아집니다.
- `beta` 값을 조절하여 정밀도와 재현율 중 더 중요하게 생각하는 지표에 가중치를 부여하는 F-beta Score도 있습니다. (F2-Score는 재현율에, F0.5-Score는 정밀도에 더 가중치를 줍니다.)

함수 인자의 종류 설명 (f1_score)

- `y_true`: 실제 레이블 (정답).
- `y_pred`: 모델의 예측 레이블.
- `labels`: (선택 사항) 계산에 포함할 레이블 목록.
- `pos_label`: (선택 사항) 이진 분류에서 양성 클래스로 간주할 레이블. 기본값은 1입니다.
- `average`: (선택 사항) 다중 클래스/다중 레이블 타겟의 경우, 평균화 방법을 지정합니다. `precision_score`와 동일합니다.
- `sample_weight`: (선택 사항) 샘플 가중치.
- `zero_division`: (선택 사항) 분모가 0일 때 반환할 값. 기본값은 'warn'이며, 0으로 나눌 때 경고를 발생시키고 0을 반환합니다.

```
import numpy as np
from sklearn.metrics import f1_score, confusion_matrix, precision_score,
recall_score

# 예시 데이터
y_true = np.array([0, 1, 0, 1, 0, 1, 0, 0, 1, 1])
y_pred = np.array([0, 1, 1, 1, 0, 0, 0, 1, 1, 1])

# scikit-learn을 이용한 F1-Score 계산
f1 = f1_score(y_true, y_pred)
print(f"F1-Score: {f1:.4f}")

# 혼동 행렬을 이용한 수동 계산
cm = confusion_matrix(y_true, y_pred)
TP = cm[1, 1]
FP = cm[0, 1]
FN = cm[1, 0]

precision = TP / (TP + FP)
recall = TP / (TP + FN)
manual_f1 = 2 * (precision * recall) / (precision + recall)
print(f"수동 계산 F1-Score: {manual_f1:.4f}")

# 다중 클래스 분류에서의 F1-Score (average 파라미터)
y_true_multi = np.array([0, 1, 2, 0, 1, 2, 0, 1, 2])
y_pred_multi = np.array([0, 1, 1, 0, 2, 2, 0, 0, 2])

# 'macro': 각 클래스별 F1-Score를 계산한 후 평균
f1_macro = f1_score(y_true_multi, y_pred_multi, average='macro')
print(f"다중 클래스 F1-Score (macro): {f1_macro:.4f}")

# 'weighted': 각 클래스별 F1-Score를 계산한 후 샘플 수에 따라 가중 평균
```

```
f1_weighted = f1_score(y_true_multi, y_pred_multi, average='weighted')
print(f"다중 클래스 F1-Score (weighted): {f1_weighted:.4f}")

# 'micro': 전체 TP, FP, FN을 합산하여 계산 (Accuracy와 동일)
f1_micro = f1_score(y_true_multi, y_pred_multi, average='micro')
print(f"다중 클래스 F1-Score (micro): {f1_micro:.4f}")

# 'None': 각 클래스별 F1-Score 반환
f1_per_class = f1_score(y_true_multi, y_pred_multi, average=None)
print(f"다중 클래스 F1-Score (클래스별): {f1_per_class}")
```

결과 해석 방법

- F1-Score 값이 1에 가까울수록 모델의 정밀도와 재현율이 모두 높고, 두 지표 간의 균형이 잘 맞는다고 해석할 수 있습니다.
- 0에 가까울수록 모델의 성능이 좋지 않음을 의미합니다.
- 불균형 데이터셋에서 모델의 성능을 평가할 때 정확도보다 더 신뢰할 수 있는 지표입니다.

장단점 및 대안

- **장점**:
  - 정밀도와 재현율의 균형을 종합적으로 평가할 수 있습니다.
  - 불균형 데이터셋에서 모델의 성능을 평가하는 데 효과적입니다.
- **단점**:
  - 정밀도와 재현율 중 어느 한쪽이 극단적으로 낮으면 F1-Score도 낮아지므로, 특정 지표가 더 중요한 상황에서는 다른 지표를 우선적으로 고려해야 할 수 있습니다.
- **대안**:
  - 특정 상황에서 정밀도나 재현율 중 하나가 더 중요하다면, F-beta Score를 사용하여 가중치를 조절할 수 있습니다.
  - `classification_report` 함수를 사용하면 Precision, Recall, F1-Score를 한 번에 확인할 수 있습니다.

ROC Curve & AUC (수신자 조작 특성 곡선 및 곡선 아래 면적)

- **ROC Curve (Receiver Operating Characteristic Curve)**: 이진 분류 모델의 성능을 시각적으로 평가하는 그래프입니다. 모델의 분류 임계값을 변경하면서 TPR(True Positive Rate, 재현율)과 FPR(False Positive Rate, 1-특이도)의 변화를 2차원 평면에 그린 곡선입니다.
  - **TPR (True Positive Rate)**: 실제 양성 중 모델이 양성으로 올바르게 예측한 비율 (재현율).
  - **FPR (False Positive Rate)**: 실제 음성 중 모델이 양성으로 잘못 예측한 비율 (1 - 특이도).
- **AUC (Area Under the Curve)**: ROC 곡선 아래의 면적을 의미합니다. 0부터 1까지의 값을 가지며, 1에 가까울수록 모델의 성능이 좋다고 평가합니다. AUC는 모델이 임의의 양성 샘플과 임의의 음성 샘플을 올바르게 분류할 확률로 해석될 수 있습니다.

적용 가능한 상황

- 이진 분류 모델의 성능을 평가하고 비교할 때 가장 널리 사용되는 지표 중 하나입니다.
- 특히 클래스 불균형이 심한 데이터셋에서 모델의 성능을 평가할 때 정확도보다 훨씬 신뢰할 수 있는 지표입니다.

- 다양한 분류 임계값에 대한 모델의 성능을 종합적으로 이해하고 싶을 때 유용합니다.
- 여러 모델의 성능을 비교하여 최적의 모델을 선택할 때 사용됩니다.

주의사항

- ROC Curve는 모델의 예측 확률(또는 결정 함수 값)을 기반으로 합니다. 따라서 모델이 예측 확률을 제공해야 합니다.
- 다중 클래스 분류에서는 각 클래스에 대해 'One-vs-Rest' 방식으로 이진 분류 문제를 적용하여 ROC Curve와 AUC를 계산할 수 있습니다.
- AUC가 높다고 해서 항상 최적의 모델인 것은 아닙니다. 특정 비즈니스 목표에 따라 정밀도나 재현율이 더 중요할 수 있으므로, 다른 지표들과 함께 고려해야 합니다.

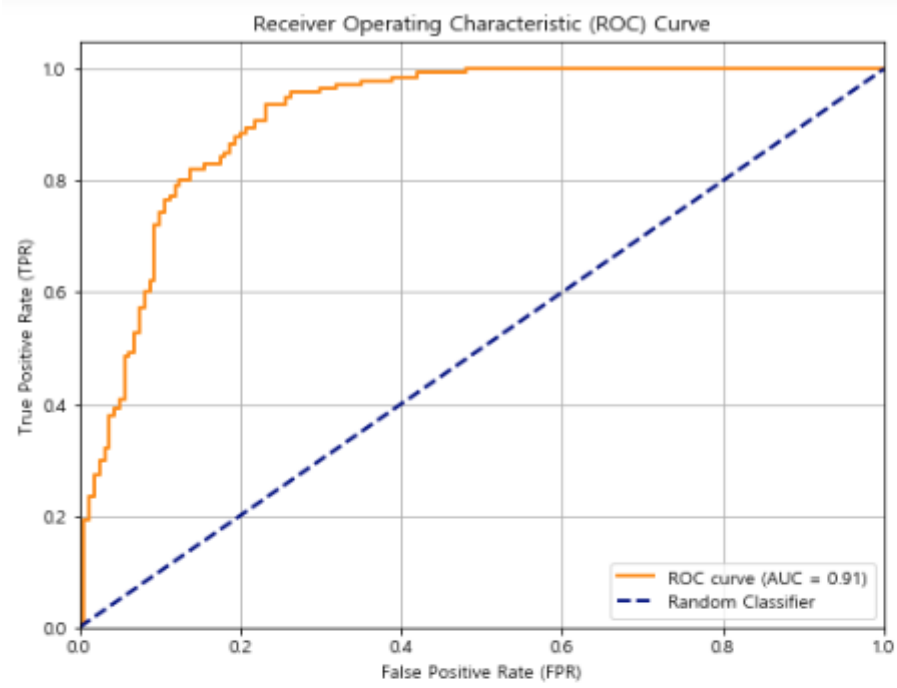
함수 인자의 종류 설명 (roc_curve, auc, roc_auc_score)

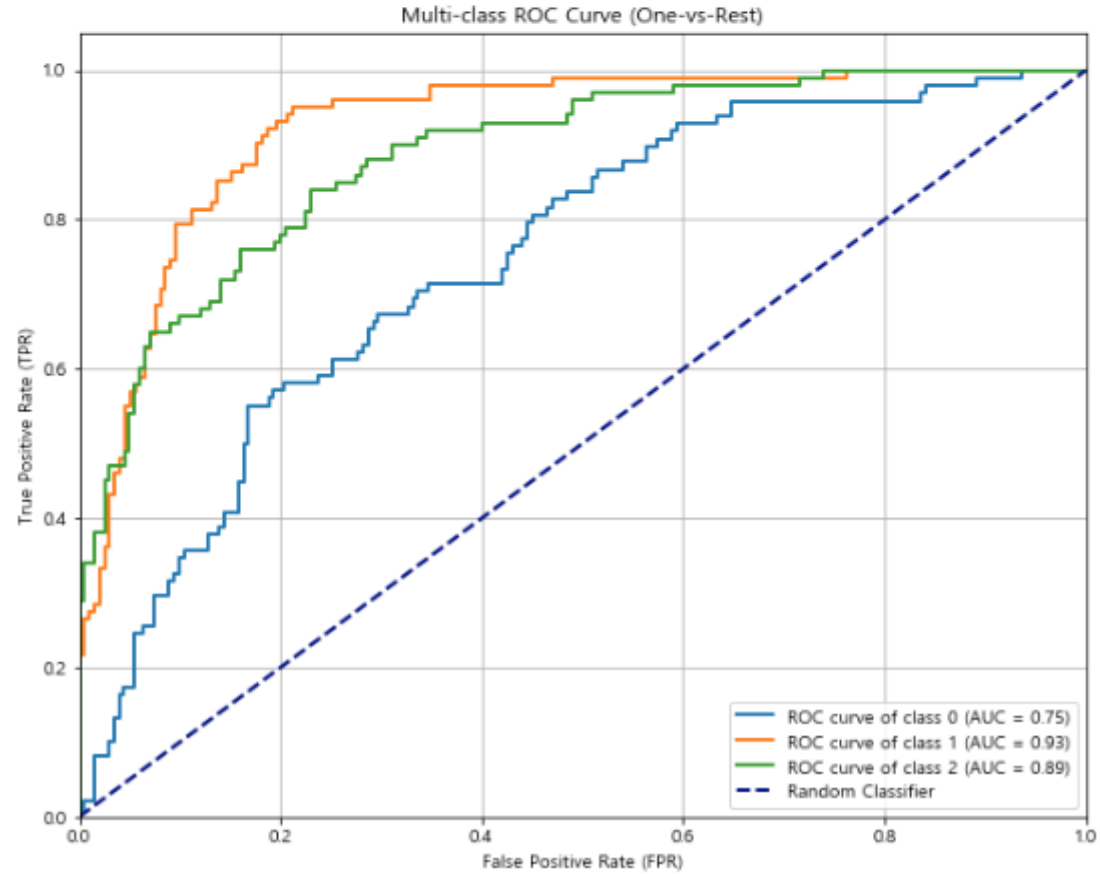
- `roc_curve(y_true, y_score, pos_label=None, sample_weight=None, drop_intermediate=True)`
  - `y_true`: 실제 이진 레이블 (0 또는 1).
  - `y_score`: 양성 클래스에 대한 예측 확률 또는 결정 함수 값.
  - `pos_label`: (선택 사항) 양성 클래스로 간주할 레이블. 기본값은 1입니다.
  - 반환 값: `fpr` (False Positive Rate), `tpr` (True Positive Rate), `thresholds` (분류 임계값).
- `auc(x, y, reorder=False)`
  - `x`: x축 값 (일반적으로 `fpr`).
  - `y`: y축 값 (일반적으로 `tpr`).
  - `reorder`: (선택 사항) x가 증가하는 순서가 아니면 재정렬할지 여부.
  - 반환 값: 곡선 아래 면적.
- `roc_auc_score(y_true, y_score, average='macro', sample_weight=None, max_fpr=None, multi_class='raise', labels=None)`
  - `y_true`: 실제 레이블.
  - `y_score`: 예측 확률 또는 결정 함수 값.
  - `average`: (선택 사항) 다중 클래스/다중 레이블 타겟의 경우, 평균화 방법을 지정합니다. `precision_score`와 유사합니다.
    - `'macro'`: 각 클래스별 AUC를 계산한 후 평균.
    - `'weighted'`: 각 클래스별 AUC를 계산한 후 샘플 수에 따라 가중 평균.
    - `'micro'`: 전체 TP, FP, FN을 합산하여 계산.
  - `multi_class`: (선택 사항) 다중 클래스 분류에서 ROC AUC를 계산하는 방법.
    - `'raise'`: 다중 클래스 입력 시 오류 발생 (기본값).
    - `'ovr'`: One-vs-Rest 전략을 사용하여 각 클래스에 대해 이진 ROC AUC를 계산한 후 평균.
    - `'ovo'`: One-vs-One 전략을 사용하여 각 클래스 쌍에 대해 이진 ROC AUC를 계산한 후 평균.
  - `labels`: (선택 사항) `multi_class`가 `'ovr'` 또는 `'ovo'`일 때 사용할 클래스 레이블 목록.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc, roc_auc_score
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.datasets import make_classification
```

```
# 예시 데이터 생성 (이진 분류)
X, y = make_classification(n_samples=1000, n_features=20, n_informative=10,
n_redundant=5, n_classes=2, random_state=42)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)

# 로지스틱 회귀 모델 학습
model = LogisticRegression(random_state=42, solver='liblinear')
model.fit(X_train, y_train)

# 예측 확률 얻기 (양성 클래스에 대한 확률)
y_pred_proba = model.predict_proba(X_test)[:, 1]

# ROC Curve 계산
fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba)

# AUC 계산
roc_auc = auc(fpr, tpr)
print(f"AUC (수동 계산): {roc_auc:.4f}")            # 0.9124

# scikit-learn을 이용한 AUC 계산
roc_auc_sklearn = roc_auc_score(y_test, y_pred_proba)
print(f"AUC (scikit-learn): {roc_auc_sklearn:.4f}") # 0.9124

# ROC Curve 시각화
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (AUC =
{roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--', label='Random
Classifier')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate (FPR)')
plt.ylabel('True Positive Rate (TPR)')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc="lower right")
plt.grid(True)
plt.show()

# 다중 클래스 분류 예시 (One-vs-Rest)
from sklearn.preprocessing import LabelBinarizer

# 예시 데이터 생성 (다중 클래스 분류)
X_multi, y_multi = make_classification(n_samples=1000, n_features=20,
n_informative=10, n_redundant=5, n_classes=3, random_state=42)
X_train_multi, X_test_multi, y_train_multi, y_test_multi =
train_test_split(X_multi, y_multi, test_size=0.3, random_state=42)

# 로지스틱 회귀 모델 학습
model_multi = LogisticRegression(random_state=42, solver='liblinear',
multi_class='ovr')
model_multi.fit(X_train_multi, y_train_multi)

# 예측 확률 얻기
```

```
y_pred_proba_multi = model_multi.predict_proba(X_test_multi)

# 각 클래스에 대한 ROC Curve 및 AUC 계산
lb = LabelBinarizer()
y_test_binarized = lb.fit_transform(y_test_multi)

plt.figure(figsize=(10, 8))
for i in range(y_multi.max() + 1):
    fpr_multi, tpr_multi, _ = roc_curve(y_test_binarized[:, i],
y_pred_proba_multi[:, i])
    roc_auc_multi = auc(fpr_multi, tpr_multi)
    plt.plot(fpr_multi, tpr_multi, lw=2, label=f'ROC curve of class {i} (AUC =
{roc_auc_multi:.2f})')

plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--', label='Random
Classifier')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate (FPR)')
plt.ylabel('True Positive Rate (TPR)')
plt.title('Multi-class ROC Curve (One-vs-Rest)')
plt.legend(loc="lower right")
plt.grid(True)
plt.show()

# 다중 클래스 AUC (macro, weighted)
auc_macro_multi = roc_auc_score(y_test_multi, y_pred_proba_multi,
multi_class='ovr', average='macro')
auc_weighted_multi = roc_auc_score(y_test_multi, y_pred_proba_multi,
multi_class='ovr', average='weighted')
print(f"다중 클래스 AUC (macro): {auc_macro_multi:.4f}")       # 0.8527
print(f"다중 클래스 AUC (weighted): {auc_weighted_multi:.4f}") # 0.8539
```





결과 해석 방법

- **ROC Curve**:
  - 곡선이 좌측 상단에 가까울수록 모델의 성능이 좋습니다. 이는 낮은 FPR에서 높은 TPR을 달성한다는 의미입니다.
  - 대각선(Random Classifier)에 가까울수록 모델의 성능이 무작위 예측과 비슷하다는 의미입니다.
- **AUC**:
  - AUC 값이 1에 가까울수록 모델이 양성 클래스와 음성 클래스를 매우 잘 구분한다고 해석할 수 있습니다.

    - AUC 값이 0.5에 가까우면 모델의 성능이 무작위 예측과 비슷하다는 의미입니다.
    - AUC 값이 0.5보다 작으면 모델이 실제보다 반대로 예측하는 경향이 있다는 의미이므로, 예측 확률을 반전시키거나 모델을 재검토해야 합니다.
    - 일반적으로 AUC 0.8 이상이면 좋은 성능으로 간주됩니다.

장단점 및 대안

- **장점**:
    - 클래스 불균형에 강하며, 모델의 전반적인 분류 성능을 평가하는 데 효과적입니다.
    - 분류 임계값에 독립적인 모델의 성능을 평가할 수 있습니다.
    - 여러 모델의 성능을 비교하는 데 유용합니다.
    - 시각적으로 모델의 성능을 이해하기 쉽습니다.
- **단점**:
    - 예측 확률이 아닌 최종 예측 레이블만 사용하는 모델에는 직접 적용하기 어렵습니다.
    - 정밀도-재현율 곡선(Precision-Recall Curve)이 특정 상황(특히 극심한 클래스 불균형)에서는 ROC Curve보다 더 유용한 정보를 제공할 수 있습니다.
- **대안**:
    - **Precision-Recall Curve**: 특히 양성 클래스의 수가 매우 적은 불균형 데이터셋에서 ROC Curve보다 더 명확한 성능 차이를 보여줄 수 있습니다.
    - `classification_report`: 다양한 평가지표를 한 번에 요약하여 보여주므로, ROC AUC와 함께 사용하여 모델 성능을 종합적으로 평가할 수 있습니다.