머신러닝 Multiclass ROC 커브 그리기

20213064_김종민

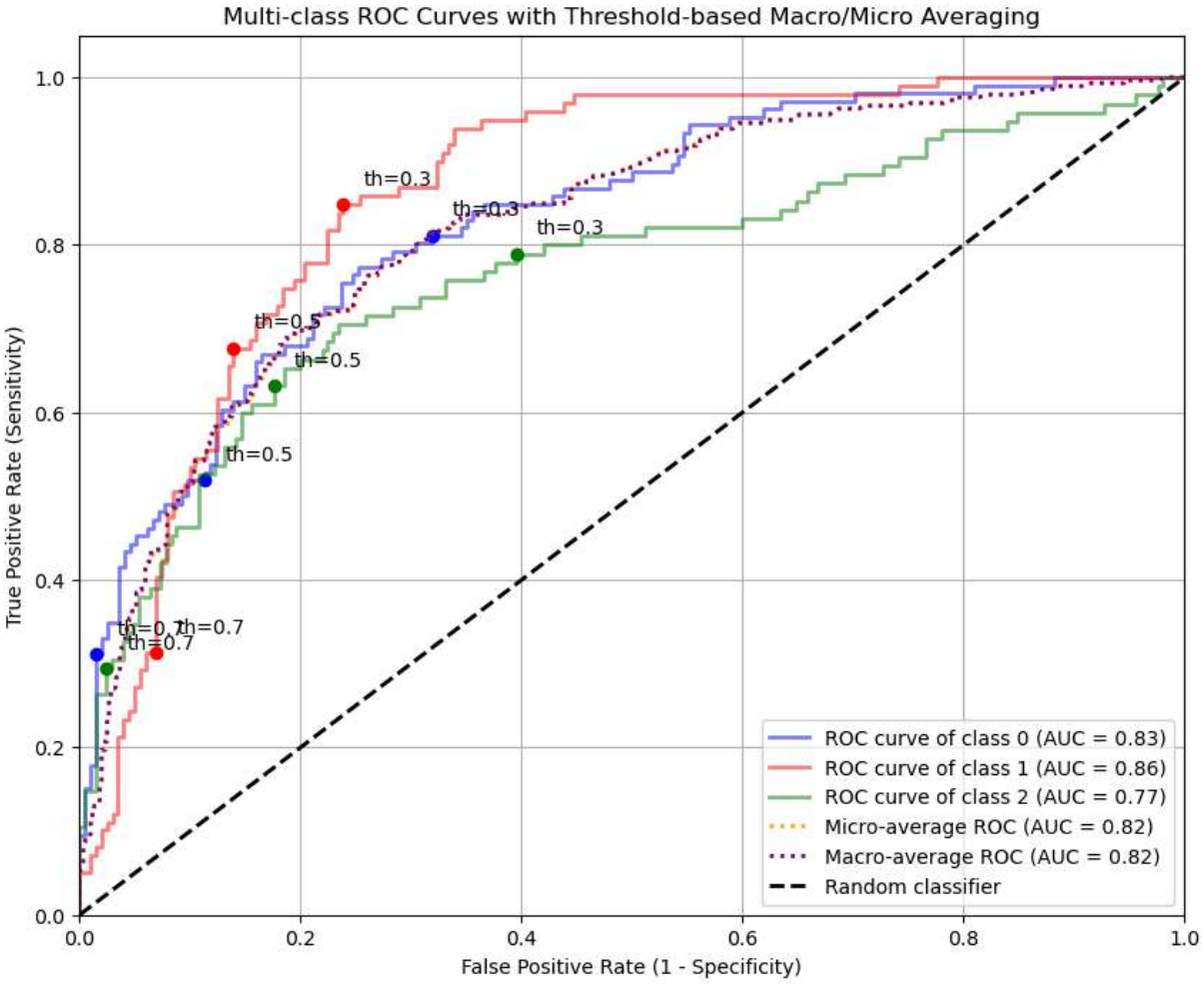
Macro Average와 Micro Average는 몇 가지 차이점이 존재합니다. Macro Average는 클래스의 가중치를 균등하게 고려합니다. 이와 달리 Micro Average는 샘플의 가중치를 균등하게 고려합니다. 즉, 특정 클래스의 샘플의 수가 많아도 Macro Average는 가중치를 클래스별로 동등하게 고려하는 반면, Micro Average는 특정 클래스의 샘플의 수가 많다면 가중치를 더 해주는 것입니다. 이는 구현하는 방식에서도 차이를 보이는데, Macro Average는 클래스별 TPR, FPR을 평균하지만 Micro Average는 모든 샘플의 TPR, FPR을 기준으로 계산합니다. 위에서도 말한 것과 같은 맥락에서 Macro Average는 클래스 크기에 따른 편향이 없어서 클래스 간 성능 차이를 동등하게 평가해야 할 때 적합하고, Micro Average는 데이터가 많은 클래스가 결과에 더 영향을 미치기 때문에 데이터 크기에 비례하여 평가해야 할 때 적합합니다.

첫 번째는 모든 클래스의 threshold값을 모아서 정렬하는 방법으로 구현했고 두 번째는 np.interp를 사용해서 클래스 사이의 값을 일정하게 보간(interpolate)하는 방법으로 구현했습니다.

구현하는 것도 구현하는 것이지만 제일 중요했다고 생각이 드는건 Macro Average와 Micro Average의 개념을 명확히 알고 있었는가였던것 같습니다.

In [10]:
```
# 모든 클래스의 threshold 값을 모아서 정렬하는 방법
import numpy as np
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_curve, auc
from sklearn.preprocessing import label_binarize
import matplotlib.pyplot as plt

X, y = make_classification(
    n_samples=1000,
    n_classes=3,
    n_features=20,
    n_informative=15,
    random_state=42
)
y = label_binarize(y, classes=[0, 1, 2])
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42
)

model = LogisticRegression(multi_class='ovr')
model.fit(X_train, np.argmax(y_train, axis=1))
y_pred_proba = model.predict_proba(X_test)

plt.figure(figsize=(10, 8))
colors = ['blue', 'red', 'green']
classes = ['Class 0', 'Class 1', 'Class 2']
all_thresholds = []
class_curves = []

# 각 클래스의 ROC 곡선과 임계값 계산
for i, color in enumerate(colors):
    fpr, tpr, thresholds = roc_curve(y_test[:, i], y_pred_proba[:, i])
    roc_auc = auc(fpr, tpr)
    class_curves.append({
        'fpr': fpr,
        'tpr': tpr,
        'thresholds': thresholds,
        'auc': roc_auc
    })
    all_thresholds.extend(thresholds)
    plt.plot(fpr, tpr, color=color, lw=2, alpha=0.5,
             label=f'ROC curve of class {i} (AUC = {roc_auc:.2f})')
    threshold_points = [0.3, 0.5, 0.7]
    for threshold in threshold_points:
        idx = np.argmin(np.abs(thresholds - threshold))
        plt.plot(fpr[idx], tpr[idx], color=color, marker='o', markersize=6)
        plt.annotate(f'th={threshold:.1f}',
                     (fpr[idx], tpr[idx]),
                     xytext=(10, 10),
                     textcoords='offset points')

unique_thresholds = np.unique(all_thresholds)
unique_thresholds = np.sort(unique_thresholds)[::-1]

# Macro-average 계산
macro_tpr = np.zeros(len(unique_thresholds))
macro_fpr = np.zeros(len(unique_thresholds))
for threshold in unique_thresholds:
    tpr_sum = 0
    fpr_sum = 0
    for i in range(3):
        y_pred_class = (y_pred_proba[:, i] >= threshold).astype(int)
        tn = np.sum((y_test[:, i] == 0) & (y_pred_class == 0))
        fp = np.sum((y_test[:, i] == 0) & (y_pred_class == 1))
        fn = np.sum((y_test[:, i] == 1) & (y_pred_class == 0))
        tp = np.sum((y_test[:, i] == 1) & (y_pred_class == 1))

        tpr = tp / (tp + fn) if (tp + fn) > 0 else 0
        fpr = fp / (fp + tn) if (fp + tn) > 0 else 0
```

```
        tpr_sum += tpr
        fpr_sum += fpr

    idx = np.where(unique_thresholds == threshold)[0][0]
    macro_tpr[idx] = tpr_sum / 3
    macro_fpr[idx] = fpr_sum / 3

# Micro-average 계산
y_test_ravel = y_test.ravel()
y_pred_proba_ravel = y_pred_proba.ravel()

micro_tpr = np.zeros(len(unique_thresholds))
micro_fpr = np.zeros(len(unique_thresholds))

for idx, threshold in enumerate(unique_thresholds):
    y_pred = (y_pred_proba_ravel >= threshold).astype(int)
    # 전체 데이터를 기준으로 True Positive, False Positive, True Negative, False Negative 계산
    tn = np.sum((y_test_ravel == 0) & (y_pred == 0))
    fp = np.sum((y_test_ravel == 0) & (y_pred == 1))
    fn = np.sum((y_test_ravel == 1) & (y_pred == 0))
    tp = np.sum((y_test_ravel == 1) & (y_pred == 1))

    # TPR (Sensitivity)와 FPR (1 - Specificity) 계산
    micro_tpr[idx] = tp / (tp + fn) if (tp + fn) > 0 else 0
    micro_fpr[idx] = fp / (fp + tn) if (fp + tn) > 0 else 0

# Macro/Micro ROC Curve
macro_auc = auc(macro_fpr, macro_tpr)
micro_auc = auc(micro_fpr, micro_tpr)

plt.plot(micro_fpr, micro_tpr, color='orange', lw=2, linestyle=':',
         label=f'Micro-average ROC (AUC = {micro_auc:.2f})')
plt.plot(macro_fpr, macro_tpr, color='purple', lw=2, linestyle=':',
         label=f'Macro-average ROC (AUC = {macro_auc:.2f})')

# Random Classifier
plt.plot([0, 1], [0, 1], color='black', lw=2, linestyle='--',
         label='Random classifier')

plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate (1 - Specificity)')
plt.ylabel('True Positive Rate (Sensitivity)')
plt.title('Multi-class ROC Curves with Threshold-based Macro/Micro Averaging')
plt.legend(loc="lower right", fontsize=10)
plt.grid(True)
plt.show()
```


Multi-class ROC Curves with Threshold-based Macro/Micro Averaging
True Positive Rate (Sensitivity)
False Positive Rate (1 - Specificity)
0.0
0.2
0.4
0.6
0.8
1.0
th=0.3
th=0.3
th=0.3
th=0.5
th=0.5
th=0.5
th=0.7
th=0.7
th=0.7
ROC curve of class 0 (AUC = 0.83)
ROC curve of class 1 (AUC = 0.86)
ROC curve of class 2 (AUC = 0.77)
Micro-average ROC (AUC = 0.82)
Macro-average ROC (AUC = 0.82)
Random classifier

In [11]:
```
# np.interp를 사용해서 클래스 사이의 값을 일정하게 보간(interpolate)하는 방법
import numpy as np
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_curve, auc
from sklearn.preprocessing import label_binarize
import matplotlib.pyplot as plt

X, y = make_classification(
    n_samples=1000,
    n_classes=3,
    n_features=20,
    n_informative=15,
    random_state=42
)
y = label_binarize(y, classes=[0, 1, 2])
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42
)
model = LogisticRegression(multi_class='ovr')
model.fit(X_train, np.argmax(y_train, axis=1))
y_pred_proba = model.predict_proba(X_test)

plt.figure(figsize=(10, 8))
colors = ['blue', 'red', 'green']
classes = ['Class 0', 'Class 1', 'Class 2']
n_classes = y.shape[1]

all_thresholds = []
class_curves = []

for i, color in enumerate(colors):
    fpr, tpr, thresholds = roc_curve(y_test[:, i], y_pred_proba[:, i])
    roc_auc = auc(fpr, tpr)
    class_curves.append({'fpr': fpr, 'tpr': tpr, 'thresholds': thresholds, 'auc': roc_auc})
    all_thresholds.extend(thresholds)
    plt.plot(fpr, tpr, color=color, lw=2, alpha=0.5,
             label=f'ROC curve of class {i} (AUC = {roc_auc:.2f})')
    threshold_points = [0.3, 0.5, 0.7]
    for threshold in threshold_points:
        idx = np.argmin(np.abs(thresholds - threshold))
        plt.plot(fpr[idx], tpr[idx], color=color, marker='o', markersize=6)
        plt.annotate(f'th={threshold:.1f}',
                     (fpr[idx], tpr[idx]),
                     xytext=(10, 10),
                     textcoords='offset points')
unique_thresholds = np.unique(all_thresholds)
unique_thresholds = np.sort(unique_thresholds)[::-1]

# Macro/Micro Average 계산
macro_tpr = []
micro_tpr = []
micro_fpr = []

for threshold in unique_thresholds:
    macro_tpr_sum = 0
    fpr_sum = 0
    tpr_sum = 0
    tn_sum, fp_sum, fn_sum, tp_sum = 0, 0, 0, 0

    for i in range(n_classes):
        # 보간으로 각 클래스의 FPR/TPR 계산
        fpr_interp = np.interp(threshold, class_curves[i]['thresholds'][::-1], class_curves[i]['fpr'][::-1])
        tpr_interp = np.interp(threshold, class_curves[i]['thresholds'][::-1], class_curves[i]['tpr'][::-1])

        macro_tpr_sum += tpr_interp

        # Micro-average를 위한 값 계산
        y_pred_class = (y_pred_proba[:, i] >= threshold).astype(int)
        tn = np.sum((y_test[:, i] == 0) & (y_pred_class == 0))
```

```
        fp = np.sum((y_test[:, i] == 0) & (y_pred_class == 1))
        fn = np.sum((y_test[:, i] == 1) & (y_pred_class == 0))
        tp = np.sum((y_test[:, i] == 1) & (y_pred_class == 1))
        tn_sum += tn
        fp_sum += fp
        fn_sum += fn
        tp_sum += tp

    macro_tpr.append(macro_tpr_sum / n_classes)
    micro_tpr.append(tp_sum / (tp_sum + fn_sum) if tp_sum + fn_sum > 0 else 0)
    micro_fpr.append(fp_sum / (fp_sum + tn_sum) if fp_sum + tn_sum > 0 else 0)

macro_auc = auc(np.array(micro_fpr), np.array(macro_tpr))
micro_auc = auc(np.array(micro_fpr), np.array(micro_tpr))

# Macro/Micro ROC Curve
plt.plot(micro_fpr, micro_tpr, color='orange', lw=2, linestyle=':',
         label=f'Micro-average ROC (AUC = {micro_auc:.2f})')
plt.plot(micro_fpr, macro_tpr, color='purple', lw=2, linestyle=':',
         label=f'Macro-average ROC (AUC = {macro_auc:.2f})')
# Random Classifier
plt.plot([0, 1], [0, 1], color='black', lw=2, linestyle='--',
         label='Random classifier')

plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate (1 - Specificity)')
plt.ylabel('True Positive Rate (Sensitivity)')
plt.title('Multi-class ROC Curves with np.interp-based Macro/Micro Averaging')
plt.legend(loc="lower right", fontsize=10)
plt.grid(True)
plt.show()
```