In [1]:
```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import label_binarize
from sklearn.linear_model import LogisticRegression
from sklearn.multiclass import OneVsRestClassifier
from sklearn.metrics import roc_curve, auc

# 데이터 생성
X, y = make_classification(
    n_samples=1000,
    n_classes=3,
    n_features=20,
    n_informative=15,
    random_state=42
)

# Train/Test split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42
)
```

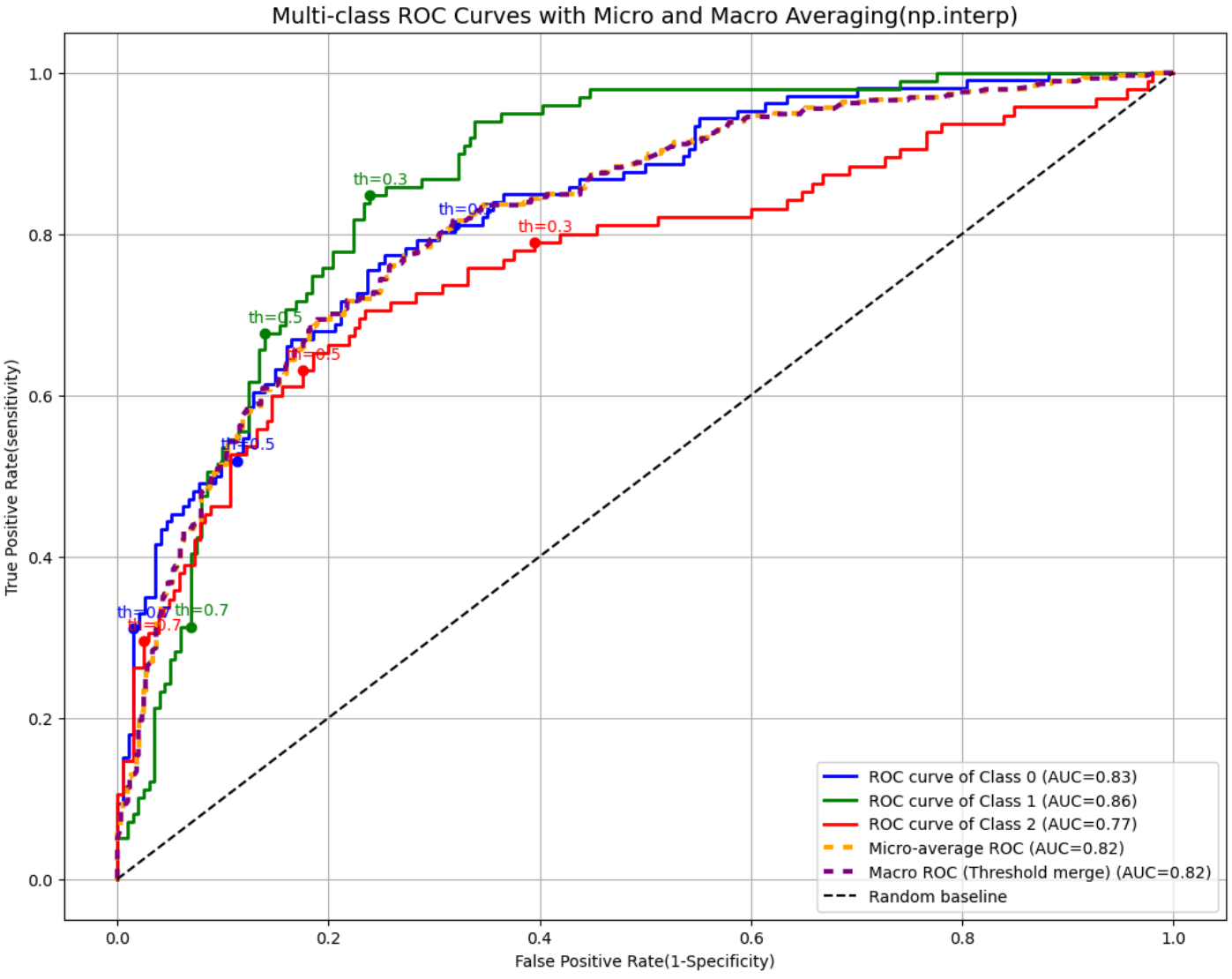
In [2]:
```python
# OVR용 원-핫 변환
y_test_bin = label_binarize(y_test, classes=[0,1,2])
n_classes = y_test_bin.shape[1]

# OVR Logistic Regression 모델 학습
model = OneVsRestClassifier(LogisticRegression(max_iter=1000))
model.fit(X_train, y_train)

# 예측 확률
y_score = model.predict_proba(X_test)

# 클래스별 ROC 계산
fpr = dict()
tpr = dict()
thresholds = dict()
roc_auc = dict()

for i in range(n_classes):
    fpr[i], tpr[i], thresholds[i] = roc_curve(y_test_bin[:, i], y_score[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])
```

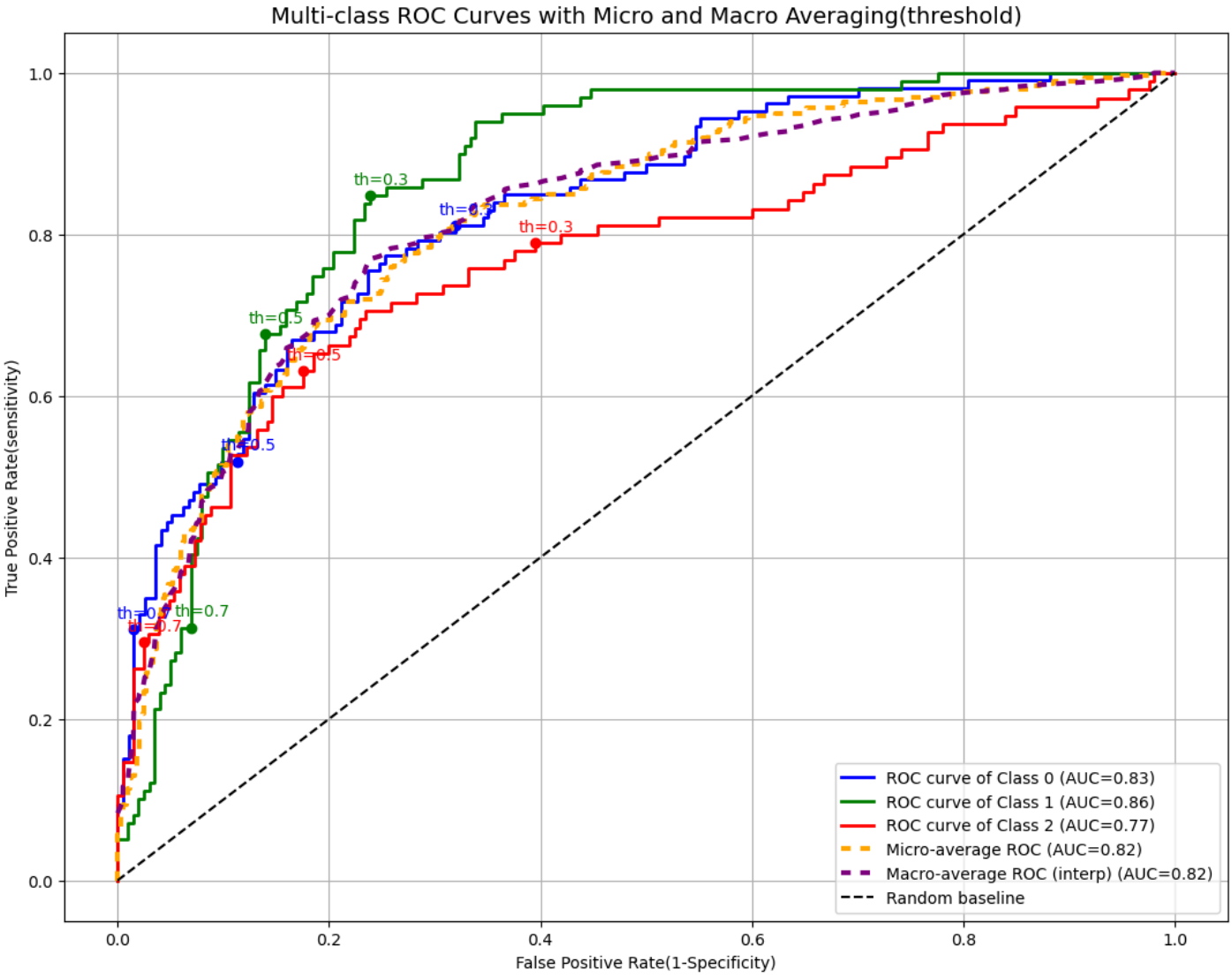
In [3]:
```python
# Micro-average ROC
fpr["micro"], tpr["micro"], _ = roc_curve(
    y_test_bin.ravel(), y_score.ravel()
)
roc_auc["micro"] = auc(fpr["micro"], tpr["micro"])



# MACRO-average ROC (threshold-based)
# 모든 threshold를 모아서 정렬
all_thresholds = np.unique(
    np.concatenate([thresholds[i] for i in range(n_classes)])
)

# 각 threshold에서 TPR/FPR을 직접 계산
tpr_macro_t = []
fpr_macro_t = []

for thr in all_thresholds:
    # threshold 기준으로 클래스별 TPR/FPR 계산
    tpr_list = []
    fpr_list = []
    for i in range(n_classes):
        pred_i = (y_score[:, i] >= thr).astype(int)
        tp = np.sum((pred_i == 1) & (y_test_bin[:, i] == 1))
        fp = np.sum((pred_i == 1) & (y_test_bin[:, i] == 0))
        fn = np.sum((pred_i == 0) & (y_test_bin[:, i] == 1))
        tn = np.sum((pred_i == 0) & (y_test_bin[:, i] == 0))

        tpr_list.append(tp / (tp + fn + 1e-10))
        fpr_list.append(fp / (fp + tn + 1e-10))
```

```
    tpr_macro_t.append(np.mean(tpr_list))
    fpr_macro_t.append(np.mean(fpr_list))

roc_auc_macro_threshold = auc(
    np.sort(fpr_macro_t),
    np.array(tpr_macro_t)[np.argsort(fpr_macro_t)]
)


# MACRO-average ROC (np.interp 보간 방식)
# 모든 FPR을 모은 뒤 정렬
all_fpr_interp = np.unique(
    np.concatenate([fpr[i] for i in range(n_classes)])
)

# FPR grid에서 TPR을 보간하여 평균
mean_tpr_interp = np.zeros_like(all_fpr_interp)

for i in range(n_classes):
    mean_tpr_interp += np.interp(all_fpr_interp, fpr[i], tpr[i])

mean_tpr_interp /= n_classes

# AUC 계산
roc_auc_macro_interp = auc(all_fpr_interp, mean_tpr_interp)
```

In [4]:
```
# 그래프 1 : 각 클래스별 ROC 커브 + Macro와 Micro average ROC 커브(모든 클래스의 threshold값을 모아서 정렬한 방법)

plt.figure(figsize=(13, 10))

# 클래스별 ROC
colors = ['blue', 'green', 'red']
threshold_points = [0.3, 0.5, 0.7]

for i, color in zip(range(n_classes), colors):
    plt.plot(fpr[i], tpr[i], lw=2, color=color,
             label=f'ROC curve of Class {i} (AUC={roc_auc[i]:.2f})')

    # 각 클래스 ROC에 threshold 점 추가
    for thr in threshold_points:
        idx = np.argmin(np.abs(thresholds[i] - thr))
        plt.plot(fpr[i][idx], tpr[i][idx], 'o', color=color)
        plt.annotate(
            f"th={thr:.1f}",
            (fpr[i][idx], tpr[i][idx]),
            textcoords="offset points",
            xytext=(7, 7),
            ha="center",
            color=color
        )

# Micro ROC
plt.plot(fpr["micro"], tpr["micro"], color='orange', linestyle=(0, (2, 2)), linewidth=3,
         label=f'Micro-average ROC (AUC={roc_auc["micro"]:.2f})')

# Macro ROC : threshold값을 모아서 정렬한 방법
plt.plot(fpr_macro_t, tpr_macro_t, color='purple', linestyle=(0, (2, 2)), linewidth=3,
         label=f'Macro ROC (Threshold merge) (AUC={roc_auc_macro_threshold:.2f})')

# Random baseline
plt.plot([0, 1], [0, 1], 'k--', label='Random baseline')

plt.title("Multi-class ROC Curves with Micro and Macro Averaging(np.interp)", fontsize=14)
plt.xlabel("False Positive Rate(1-Specificity)")
plt.ylabel("True Positive Rate(sensitivity)")
plt.legend(loc="lower right", fontsize=10)
plt.grid(True)
plt.show()


# 그래프 2 : 각 클래스별 ROC 커브 + Macro와 Micro average ROC 커브(np.interp를 사용)

plt.figure(figsize=(13, 10))

# 클래스별 ROC
colors = ['blue', 'green', 'red']
```

```python
threshold_points = [0.3, 0.5, 0.7]

for i, color in zip(range(n_classes), colors):
    plt.plot(fpr[i], tpr[i], lw=2, color=color,
             label=f'ROC curve of Class {i} (AUC={roc_auc[i]:.2f})')

    # 각 클래스 ROC에 threshold 점 추가
    for thr in threshold_points:
        idx = np.argmin(np.abs(thresholds[i] - thr))
        plt.plot(fpr[i][idx], tpr[i][idx], 'o', color=color)
        plt.annotate(
            f"th={thr:.1f}",
            (fpr[i][idx], tpr[i][idx]),
            textcoords="offset points",
            xytext=(7, 7),
            ha="center",
            color=color
        )

# Micro ROC
plt.plot(fpr["micro"], tpr["micro"], color='orange', linestyle=(0, (2, 2)), linewidth=3,
         label=f'Micro-average ROC (AUC={roc_auc["micro"]:.2f})')

# Macro ROC : np.interp를 사용해서 클래스 사이의 값을 일정하게 보간한 방법
plt.plot(all_fpr_interp, mean_tpr_interp, color='purple', linestyle=(0, (2, 2)), linewidth=3,
         label=f'Macro-average ROC (interp) (AUC={roc_auc_macro_interp:.2f})')

# Random baseline
plt.plot([0, 1], [0, 1], 'k--', label='Random baseline')

plt.title("Multi-class ROC Curves with Micro and Macro Averaging(threshold)", fontsize=14)
plt.xlabel("False Positive Rate(1-Specificity)")
plt.ylabel("True Positive Rate(sensitivity)")
plt.legend(loc="lower right", fontsize=10)
plt.grid(True)
plt.show()
```




Multi-class ROC Curves with Micro and Macro Averaging(threshold)
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=0.5
th=0.5
th=0.3
th=0.5
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 (interp) (AUC=0.82)
Random baseline