Multilabel_Roc_Auprc

Professor Elaine Cecília Gatto - Cissa

Summary

MULTILABEL AUPRC AND ROC CURVES

If you identify any errors in this material - whether conceptual, typographical, or otherwise - please contact us. Your collaboration is very welcome: elainececiliagatto@gmail.com

This material and code were developed by Professor Elaine Cecília Gatto (Cissa), a professor in the Department of Applied Computing (DAC) at the Institute of Exact and Earth Sciences (ICET) of the Federal University of Lavras (UFLA).

The development benefited from the guidance and collaboration of Professors Ricardo Cerri, from the Institute of Mathematical and Computer Sciences (ICMC) of the University of São Paulo (USP), São Carlos Campus, and Mauri Ferrandin, from the Department of Control, Automation and Computer Engineering of the Federal University of Santa Catarina (UFSC), Blumenau Campus.

1. Multilabel Cunfusion Matrix

Positive condition (P): the number of actual positive cases in the data, that is, the instance is originally positive for the class;

Negative condition (N): the number of actual negative cases in the data, that is, the instance is originally negative for the class;

True Positive (TP): number of instances of the positive class correctly classified, that is, the class is 1 and the model predicted 1;

True Negative (TN): number of instances of the negative class correctly classified, that is, the class is 0 and the model predicted 0;

False Positive (FP): number of instances where the true class is negative but were incorrectly classified, that is, as belonging to the positive class (the class is 0 but the model predicted 1);

False Negative (FN): number of instances originally belonging to the positive class that were incorrectly predicted, that is, as belonging to the negative class (the class is 1 but the model predicted 0).

		Predicted condition		Sources: [16][17][18][19][20][21][22][23] view·talk·edit	
	Total population $= P + N$	**Predicted positive**	**Predicted negative**	Informedness, bookmaker informedness (BM) $= TPR + TNR - 1$	Prevalence threshold (PT) $= \frac{\sqrt{TPR \times FPR} - FPR}{TPR - FPR}$
Actual condition	**Real Positive (P)** [a]	**True positive** (TP), hit[b]	**False negative** (FN), miss, underestimation	True positive rate (TPR), recall, sensitivity (SEN), probability of detection, hit rate, power $= \frac{TP}{P} = 1 - FNR$	False negative rate (FNR), miss rate type II error [c] $= \frac{FN}{P} = 1 - TPR$
	Real Negative (N)[d]	**False positive** (FP), false alarm, overestimation	**True negative** (TN), correct rejection[e]	False positive rate (FPR), probability of false alarm, fall-out type I error [f] $= \frac{FP}{N} = 1 - TNR$	True negative rate (TNR), specificity (SPC), selectivity $= \frac{TN}{N} = 1 - FPR$
	Prevalence $= \frac{P}{P + N}$	Positive predictive value (PPV), precision $= \frac{TP}{TP + FP} = 1 - FDR$	False omission rate (FOR) $= \frac{FN}{TN + FN} = 1 - NPV$	Positive likelihood ratio (LR+) $= \frac{TPR}{FPR}$	Negative likelihood ratio (LR−) $= \frac{FNR}{TNR}$
	Accuracy (ACC) $= \frac{TP + TN}{P + N}$	False discovery rate (FDR) $= \frac{FP}{TP + FP} = 1 - PPV$	Negative predictive value (NPV) $= \frac{TN}{TN + FN} = 1 - FOR$	Markedness (MK), deltaP (Δp) $= PPV + NPV - 1$	Diagnostic odds ratio (DOR) $= \frac{LR+}{LR-}$
	Balanced accuracy (BA) $= \frac{TPR + TNR}{2}$	F_1 score $= \frac{2\,PPV \times TPR}{PPV + TPR} = \frac{2\,TP}{2\,TP + FP + FN}$	Fowlkes–Mallows index (FM) $= \sqrt{PPV \times TPR}$	*phi* or Matthews correlation coefficient (MCC) $= \sqrt{TPR \times TNR \times PPV \times NPV} - \sqrt{FNR \times FPR \times FOR \times FDR}$	Threat score (TS), critical success index (CSI), Jaccard index $= \frac{TP}{TP + FN + FP}$

Figure 1: Figure 1: Matrix Confusion (from wikipedia)

1.1. Computing Step-by-Step

Open csvs:

```
true = pd.read_csv("~/AuprcRoc/data-real/GnegativeGO-results-lcc/Split-1/y_true.csv")
pred = pd.read_csv("~/AuprcRoc/data-real/GnegativeGO-results-lcc/Split-1/y_proba.csv")
```

True Labels (Y)

```
print(true)
```

```
     Label1  Label2  Label3  Label4  Label5  Label6  Label7  Label8
0         1       0       0       0       0       0       0       0
1         1       0       0       0       0       0       0       0
2         1       0       0       0       0       0       0       0
3         1       0       0       0       0       0       0       0
4         1       0       0       0       0       0       0       0
..      ...     ...     ...     ...     ...     ...     ...     ...
135       0       0       0       0       0       0       0       1
136       0       0       0       0       0       0       0       1
137       0       0       0       0       0       0       0       1
138       0       0       0       0       0       0       0       1
139       0       0       0       0       0       0       0       1

[140 rows x 8 columns]
```

Predict Labels ($\hat{Y}$)

```
print(pred)
```

```
     Label1  Label2  Label3  Label4  Label5  Label6  Label7  Label8
0     0.890   0.000   0.065   0.000   0.005   0.000     0.0  0.0250
1     0.895   0.035   0.070   0.000   0.000   0.000     0.0  0.0050
2     0.995   0.000   0.000   0.005   0.000   0.005     0.0  0.0000
3     1.000   0.000   0.000   0.000   0.000   0.000     0.0  0.0000
4     0.980   0.000   0.020   0.000   0.000   0.000     0.0  0.0100
..      ...     ...     ...     ...     ...     ...     ...     ...
135   0.010   0.000   0.045   0.005   0.000   0.000     0.0  0.8600
136   0.000   0.000   0.000   0.000   0.000   0.000     0.0  0.9950
137   0.000   0.000   0.005   0.005   0.000   0.000     0.0  0.9850
138   0.015   0.215   0.065   0.015   0.000   0.000     0.0  0.6745
139   0.045   0.100   0.095   0.005   0.005   0.000     0.0  0.6945

[140 rows x 8 columns]
```

Calculate the total number of instances.

```
print(len(true))
print(len(pred))
```

```
140
140
```

Calculate the total number of labels.

```
print(len(true.columns))
print(len(pred.columns))
```

```
8
8
```

Calculate the number of positive and negative instances.

```
def count_labels(df):
    """
    Counts the total number of 0s and 1s in an array, list,
    or Series of binary values.

    Parameters:
    -----------
    data : pd.DataFrame
    Values containing 0s and 1s.

    Returns:
    --------
    pd.DataFrame : table with the counts of zeros and ones.
    """
    counts = pd.DataFrame({
        'zeros': (df == 0).sum(),
        'ones': (df == 1).sum()
    })
    return counts
```

True labels (Y)

```
count_labels(true)
```

	zeros	ones
Label1	84	56
Label2	128	12
Label3	99	41

	zeros	ones
Label4	126	14
Label5	137	3
Label6	138	2
Label7	139	1
Label8	122	18

Predict labels ($\hat{Y}$)

```
count_labels(pred)
```

	zeros	ones
Label1	27	25
Label2	91	1
Label3	43	7
Label4	83	2
Label5	131	3
Label6	135	0
Label7	123	0
Label8	61	2

Now that we have the predicted labels $\hat{Y}$ and the true labels Y loaded, we need to confirm some information and save it for later use. What we need to do now is confirm when, in each instance, the true labels are true or false, and the predicted labels are true or false, that is:

$$\text{if} \begin{cases} Y = 1 & \text{then } Y' = 1 \quad \text{otherwise } Y' = 0 \\ Y = 0 & \text{then } Y' = 1 \quad \text{otherwise } Y' = 0 \\ \hat{Y} = 1 & \text{then } \hat{Y}' = 1 \quad \text{otherwise } \hat{Y}' = 0 \\ \hat{Y} = 0 & \text{then } \hat{Y}' = 1 \quad \text{otherwise } \hat{Y}' = 0 \end{cases}$$

where Y' (true) and $\hat{Y}'$ (pred) are the resulting values corresponding to Y and $\hat{Y}$. In Python we can do the following.

Computing $Y == 1$?

If $Y == 1$ then $Y' = 1$, otherwise $Y' = 0$. True label is equal to 1?. Each position receives 1 if the corresponding value in true data frame is 1, and 0 otherwise.

```
true_1 = pd.DataFrame(
    np.where(true == 1, 1, 0),
    index=pred.index,
    columns=pred.columns
)
print(true_1)
```

```
     Label1  Label2  Label3  Label4  Label5  Label6  Label7  Label8
0         1       0       0       0       0       0       0       0
1         1       0       0       0       0       0       0       0
2         1       0       0       0       0       0       0       0
3         1       0       0       0       0       0       0       0
4         1       0       0       0       0       0       0       0
..      ...     ...     ...     ...     ...     ...     ...     ...
135       0       0       0       0       0       0       0       1
136       0       0       0       0       0       0       0       1
137       0       0       0       0       0       0       0       1
138       0       0       0       0       0       0       0       1
139       0       0       0       0       0       0       0       1

[140 rows x 8 columns]
```

Computing $Y' == 0$?

if $Y == 0$ then $Y' = 1$, otherwise $Y' = 0$. True label is equal to 0?. Each position receives 1 if the corresponding value in true dataframe is 0, and 0 otherwise.

```
true_0 = pd.DataFrame(
    np.where(true == 0, 1, 0),
    index=pred.index,
    columns=pred.columns
)
print(true_0)
```

```
     Label1  Label2  Label3  Label4  Label5  Label6  Label7  Label8
0         0       1       1       1       1       1       1       1
1         0       1       1       1       1       1       1       1
2         0       1       1       1       1       1       1       1
3         0       1       1       1       1       1       1       1
4         0       1       1       1       1       1       1       1
..      ...     ...     ...     ...     ...     ...     ...     ...
135       1       1       1       1       1       1       1       0
136       1       1       1       1       1       1       1       0
137       1       1       1       1       1       1       1       0
138       1       1       1       1       1       1       1       0
139       1       1       1       1       1       1       1       0

[140 rows x 8 columns]
```

Computing $\hat{Y}' == 1$?

If $\hat{Y} == 1$ then $\hat{Y}' = 1$, otherwise $\hat{Y}' = 0$. Predict label is equal to 1?. Each position receives 1 if the corresponding value in pred dataframe is 1, and 0 otherwise.

```
pred_1 = pd.DataFrame(
    np.where(pred == 1, 1, 0),
    index=pred.index,
    columns=pred.columns
)
print(pred_1)
```

```
     Label1  Label2  Label3  Label4  Label5  Label6  Label7  Label8
0         0       0       0       0       0       0       0       0
1         0       0       0       0       0       0       0       0
2         0       0       0       0       0       0       0       0
3         1       0       0       0       0       0       0       0
4         0       0       0       0       0       0       0       0
..      ...     ...     ...     ...     ...     ...     ...     ...
135       0       0       0       0       0       0       0       0
136       0       0       0       0       0       0       0       0
137       0       0       0       0       0       0       0       0
138       0       0       0       0       0       0       0       0
139       0       0       0       0       0       0       0       0

[140 rows x 8 columns]
```

Computing $\hat{Y} == 0$?

if $\hat{Y} == 0$ then $\hat{Y}' = 1$, otherwise $\hat{Y}' = 0$. Predict label is equal to 0? Each position receives 1 if the corresponding value in pred dataframe is 0, and 0 otherwise.

```
pred_0 = pd.DataFrame(
    np.where(pred == 0, 1, 0),
    index=pred.index,
    columns=pred.columns
)
print(pred_0)
```

```
     Label1  Label2  Label3  Label4  Label5  Label6  Label7  Label8
0         0       1       0       1       0       1       1       0
1         0       0       0       1       1       1       1       0
2         0       1       1       0       1       0       1       1
3         0       1       1       1       1       1       1       1
4         0       1       0       1       1       1       1       0
..      ...     ...     ...     ...     ...     ...     ...     ...
135       0       1       0       0       1       1       1       0
136       1       1       1       1       1       1       1       0
137       1       1       0       0       1       1       1       0
138       0       0       0       0       1       1       1       0
139       0       0       0       0       0       1       1       0

[140 rows x 8 columns]
```

We can also calculate the total for each label:

Total true_0

```
total_true_1 = pd.DataFrame(true_1.sum(axis=0), columns=['sum'])
print(total_true_1)
```

```
        sum
Label1   56
Label2   12
Label3   41
Label4   14
Label5    3
Label6    2
Label7    1
Label8   18
```

Total true_1

```
total_true_0 = pd.DataFrame(true_0.sum(axis=0), columns=['sum'])
print(total_true_0)
```

```
        sum
Label1   84
Label2  128
Label3   99
Label4  126
Label5  137
Label6  138
Label7  139
Label8  122
```

Total pred_1

```
total_pred_1 = pd.DataFrame(pred_1.sum(axis=0), columns=['sum'])
print(total_pred_1)
```

```
        sum
Label1   25
Label2    1
Label3    7
Label4    2
Label5    3
Label6    0
Label7    0
Label8    2
```

Total pred_0

```
total_pred_0 = pd.DataFrame(pred_0.sum(axis=0), columns=['sum'])
print(total_pred_0)
```

```
        sum
Label1   27
Label2   91
Label3   43
Label4   83
Label5  131
Label6  135
Label7  123
Label8   61
```

Data frame with all

```
matrix_totals = pd.concat(
    [total_true_0, total_true_1, total_pred_0, total_pred_1],
    axis=1
)
matrix_totals.columns = ["total_true_0", "total_true_1",
"total_pred_0", "total_pred_1"]
print(matrix_totals)
```

```
        total_true_0  total_true_1  total_pred_0  total_pred_1
Label1            84            56            27            25
Label2           128            12            91             1
Label3            99            41            43             7
Label4           126            14            83             2
Label5           137             3           131             3
Label6           138             2           135             0
Label7           139             1           123             0
Label8           122            18            61             2
```

Now that we have the values of Y' and $\hat{Y}'$, we can calculate the number of true positives, true negatives, false positives, and false negatives, which make up the confusion matrix. Let's consider that the positive class is 1 and the negative class is 0. We can summarize as follows:

$$\text{if} \begin{cases} TPi = ((Y' == 1)\text{AND}(\hat{Y}' == 1)) & \text{then } 1 \quad \text{otherwise } 0 \\ TNi = ((Y' == 0)\text{AND}(\hat{Y}' == 0)) & \text{then } 1 \quad \text{otherwise } 0 \\ FPi = ((Y' == 0)\text{AND}(\hat{Y}' == 1)) & \text{then } 1 \quad \text{otherwise } 0 \\ FNi = ((Y' == 1)\text{AND}(\hat{Y}' == 0)) & \text{then } 1 \quad \text{otherwise } 0 \end{cases}$$

The following figure presents several ways to represent the confusion matrix:

- *True Positive*: The model predicted 1 and the correct answer is 1

```
TPi = np.where((true_1 & pred_1), 1, 0)
TPi
```

```
array([[0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       ...,
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0]], shape=(140, 8))
```

- *True Negative*: The model predicted 0 and the correct answer is 0

```
TNi = np.where((true_0 & pred_0), 1, 0)
TNi
```

```
array([[0, 1, 0, ..., 1, 1, 0],
       [0, 0, 0, ..., 1, 1, 0],
       [0, 1, 1, ..., 0, 1, 1],
       ...,
       [1, 1, 0, ..., 1, 1, 0],
       [0, 0, 0, ..., 1, 1, 0],
       [0, 0, 0, ..., 1, 1, 0]], shape=(140, 8))
```

- *False Positive*: The model predicted 1 and the correct answer is 0.

```
FPi = np.where((true_0 & pred_1), 1, 0)
FPi
```

```
array([[0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       ...,
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0]], shape=(140, 8))
```

- *False Negative*: The model predicted 0 and the correct answer is 1

```
FNi = np.where((true_1 & pred_0), 1, 0)
FNi
```

```
array([[0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       ...,
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0]], shape=(140, 8))
```

Now let's calculate the totals for each label:

$$TP_l = \sum_{i=1}^{m} TP_i$$

$$TN_l = \sum_{i=1}^{m} TN_i$$

$$FP_l = \sum_{i=1}^{m} FP_i$$

$$FN_l = \sum_{i=1}^{m} FN_i$$

- Total of true positives

```
TPl = TPi.sum(axis=0)
TPl
```

```
array([25,  1,  7,  2,  3,  0,  0,  2])
```

- Total of true negatives

```
TNl = TNi.sum(axis=0)
TNl
```

```
array([ 27,  91,  43,  83, 131, 135, 123,  61])
```

- Total of false positives

```
FPl = FPi.sum(axis=0)
FPl
```

```
array([0, 0, 0, 0, 0, 0, 0, 0])
```

- Total of false negatives

```
FNl = FNi.sum(axis=0)
FNl
```

```
array([0, 0, 0, 0, 0, 0, 0, 0])
```

- Concatenating

```
matrix_confusion_per_labels = pd.DataFrame({
    "TP": TPl,
    "TN": TNl,
    "FP": FPl,
    "FN": FNl
})
print(matrix_confusion_per_labels)
```

```
   TP   TN  FP  FN
0  25   27   0   0
1   1   91   0   0
2   7   43   0   0
3   2   83   0   0
4   3  131   0   0
5   0  135   0   0
6   0  123   0   0
7   2   61   0   0
```

We can also calculate the percentage for each label by dividing the values by the total number of instances:

```
matrix_confusion_per_labels_percent = (matrix_confusion_per_labels / len(true)) * 100
print(matrix_confusion_per_labels_percent.round(2))
```

```
      TP     TN   FP   FN
0  17.86  19.29  0.0  0.0
1   0.71  65.00  0.0  0.0
2   5.00  30.71  0.0  0.0
3   1.43  59.29  0.0  0.0
4   2.14  93.57  0.0  0.0
5   0.00  96.43  0.0  0.0
6   0.00  87.86  0.0  0.0
7   1.43  43.57  0.0  0.0
```

Now we can calculate the number of correct and incorrect answers (as well as the percentage) and combine all the information into a single table.

- Calculating the total number of correct classified labels.

```
correct = matrix_confusion_per_labels['TP'] + matrix_confusion_per_labels['TN']
correct
```

```
0     52
1     92
2     50
3     85
4    134
5    135
6    123
7     63
dtype: int64
```

- Calculating the total number of wrong classified labels.

```
wrong = matrix_confusion_per_labels['FP'] + matrix_confusion_per_labels['FN']
wrong
```

```
0    0
1    0
2    0
3    0
4    0
5    0
6    0
7    0
dtype: int64
```

1.2. More optimized calculation

Or we can do something simpler and more direct, as shown below!

- Check that the two Data Frames have the same columns and size.

```
assert list(true.columns) == list(pred.columns), "As colunas de y_true e y_pred devem ser idênticas!
assert true.shape == pred.shape, "Os DataFrames devem ter o mesmo formato!"
```

- Function to calculate TP, TN, FP, FN per column

```
def confusion_per_label(y_true_col, y_pred_col):
    arr_y_true = np.array(y_true_col)
    arr_y_pred = np.array(y_pred_col)
    TP = int(((arr_y_true == 1) & (arr_y_pred == 1)).sum())
    TN = int(((arr_y_true == 0) & (arr_y_pred == 0)).sum())
    FP = int(((arr_y_true == 0) & (arr_y_pred == 1)).sum())
    FN = int(((arr_y_true == 1) & (arr_y_pred == 0)).sum())
    return TP, TN, FP, FN
```

- Concatenating and computing the percentage for each label by dividing the values by the total number of instances:

```
results = []
m = len(true)  # total de instâncias por rótulo

for col in true.columns:
    TP, TN, FP, FN = confusion_per_label(true[col], pred[col])

    certos = TP + TN
    errados = FP + FN

    percentCertos = certos / m * 100
    percentErrados = errados / m * 100

    results.append({
        "Class": col,
        "TP": TP,
        "TN": TN,
        "FP": FP,
        "FN": FN,
        "Correct": certos,
        "Wrong": errados,
        "%Correct": percentCertos,
        "%Wrong": percentErrados
    })
matriz_confusao_1 = pd.DataFrame(results)
print(matriz_confusao_1.round(2).to_string(index=False))
```

```
 Class  TP  TN  FP  FN  Correct  Wrong  %Correct  %Wrong
Label1  25  27   0   0       52      0     37.14     0.0
Label2   1  91   0   0       92      0     65.71     0.0
Label3   7  43   0   0       50      0     35.71     0.0
Label4   2  83   0   0       85      0     60.71     0.0
Label5   3 131   0   0      134      0     95.71     0.0
Label6   0 135   0   0      135      0     96.43     0.0
Label7   0 123   0   0      123      0     87.86     0.0
Label8   2  61   0   0       63      0     45.00     0.0
```

2. Computing metrics from confusion matrix

True positive rate (TPR or Recall): probability of detection, or sensibility. Measures the ability of the test/model to correctly detect positives (true positives among all actual positives).

$$TPR(recall - sensibility) = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

True negative rate (TNR): specificity. It measures the ability of the test/model to correctly identify the negatives (true negatives among all actual negatives).

$$TNR(specificity) = \frac{\text{TN}}{\text{TN} + \text{FP}}$$

False Positive rate (FPR): probability of false alarm. The proportion of true negatives that were incorrectly classified as positives.

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

False Negative rate (FNR): miss rate. The proportion of true positives that were incorrectly classified as negatives.

$$FNR = \frac{\text{FN}}{\text{FN} + \text{TP}}$$

```
results = []
m = len(true)

for col in true.columns:
    TP, TN, FP, FN = confusion_per_label(true[col], pred[col])
    sensitivity = TP / (TP + FN) if (TP + FN) > 0 else 0
    specificity = TN / (TN + FP) if (TN + FP) > 0 else 0
    fpr = FP / (FP + TN) if (FP + TN) > 0 else 0
    fnr = FN / (FN + TP) if (FN + TP) > 0 else 0
    P = TP + FN
    N = TN + FP

    results.append({
        "Class": col,
        "P": round(P, 3),
        "N": round(N, 3),
        "TP": TP,
        "TN": TN,
        "FP": FP,
        "FN": FN,
        "TPR": round(sensitivity * 100, 2),
        "TNR": round(specificity * 100, 2),
        "FPR": round(fpr * 100, 2),
        "FNR": round(fnr * 100, 2)
    })
matriz_confusao_2 = pd.DataFrame(results)
print(matriz_confusao_2.round(2).to_string(index=False))
```

```
 Class  P   N  TP  TN  FP  FN   TPR   TNR  FPR  FNR
Label1 25  27  25  27   0   0 100.0 100.0  0.0  0.0
```

```
Label2  1  91   1  91   0   0 100.0 100.0  0.0  0.0
Label3  7  43   7  43   0   0 100.0 100.0  0.0  0.0
Label4  2  83   2  83   0   0 100.0 100.0  0.0  0.0
Label5  3 131   3 131   0   0 100.0 100.0  0.0  0.0
Label6  0 135   0 135   0   0   0.0 100.0  0.0  0.0
Label7  0 123   0 123   0   0   0.0 100.0  0.0  0.0
Label8  2  61   2  61   0   0 100.0 100.0  0.0  0.0
```

Informedness (BM): also called Bookmaker Informedness or Youden's J statistic, measures how well the classifier is informed about the distinction between classes.

$$BM = TPR + TNR - 1$$

Prevalence: it measures the proportion of true positive cases in the total sample. It represents how frequent the positive class is within the dataset.

$$Prevalence = \frac{P}{P+N}$$

Prevalence Threshold (PT): a metric derived from ROC curves that represents the point at which the benefits and costs of predicting positive or negative are balanced.

$$PT = \frac{\sqrt{TPR \times FPR} - FPR}{TPR - FPR}$$

Positive likelihood ratio (LR+): it measures how much more likely a positive result is when the true class is positive compared to when the true class is negative.

$$LR+ = \frac{TPR}{PR}$$

Negative likelihood ratio (LR-): it measures how much more likely a negative result is when the true class is positive compared to when the true class is negative.

$$LR- = \frac{FNR}{TNR}$$

Diagnostic odds ratio (DOR): it measures the overall effectiveness of a test or model in distinguishing between positive and negative true classes. It represents the ratio of the odds of a positive result in true positives to the odds of a positive result in true negatives.

$$DOR = \frac{LR+}{LR-}$$

False discovery rate (FDR): it measures the proportion of positive predictions that are actually incorrect. It indicates how often a positive prediction corresponds to a false positive.

$$FDR = \frac{FP}{TP + FP} = 1 - \text{PPV}$$

Positive predictive value (PPV or Precision): it measures the proportion of positive predictions that are truly positive. It indicates how reliable a positive prediction is.

$$PVV(precision) = \frac{TP}{TP + FP} = 1 - FDR$$

Negative predictive value (NPV): it measures the proportion of negative predictions that are actually correct. It indicates how often a negative prediction corresponds to a true negative.

$$NPV = \frac{TN}{TN + FN} = 1 - \text{FOR}$$

False omission rate (FOR): it measures the proportion of negative predictions that are actually positive. It indicates how often a negative prediction is incorrect.

$$FOR = \frac{FN}{TN + FN} = 1 - NPV$$

Accuracy (ACC): it measures the overall proportion of correct predictions (both positive and negative) among all evaluated cases. It indicates how often the model makes the correct prediction.

$$ACC = \frac{TP + TN}{P + N} = \frac{TP + TN}{TP + TN + FP + FN}$$

Balanced accuracy (BA): it measures the average of the true positive rate (sensitivity) and the true negative rate (specificity). It balances the model's performance on both positive and negative classes, making it more reliable when classes are imbalanced.

$$BA = \frac{\text{TPR} + \text{TNR}}{2}$$

F1 score: it measures the harmonic mean between precision (PPV) and recall (TPR). It balances both false positives and false negatives, providing a single score that reflects overall classification performance on the positive class.

$$F1 = \frac{2 \times \text{PPV} \times \text{TPR}}{\text{PPV} + \text{TPR}} = \frac{2 \times TP}{2TP + FP + FN}$$

Fowlkes–Mallows index (FM): it measures the geometric mean between precision (PPV) and recall (TPR). It indicates how well the model balances true positives relative to both false positives and false negatives.

$$FM = \sqrt{\text{PPV} \times \text{TPR}}$$

Matthews Correlation Coefficient (MCC): it measures the overall quality of binary classifications by considering all parts of the confusion matrix. It behaves like a correlation coefficient between true and predicted labels, producing values from -1 (total disagreement) to $+1$ (perfect prediction).

$$\phi \text{ (MCC)} = \sqrt{TPR \times TNR \times PPV \times NPV} - \sqrt{FNR \times FPR \times FOR \times FDR}$$

Threat score (TS): also known as the Critical Success Index (CSI) or Jaccard Index — measures the proportion of correctly predicted positive cases (true positives) out of all instances where a positive condition was either predicted or actually present. It provides a balanced evaluation of performance by penalizing both false alarms (false positives) and misses (false negatives), making it particularly useful for rare event prediction tasks such as weather forecasting, fault detection, or medical diagnosis.

$$TS = CSI = J = \frac{TP}{TP + FN + FP}$$

```
results_3 = []
m = len(true)

for col in true.columns:
    # Basics
    TP, TN, FP, FN = confusion_per_label(true[col], pred[col])
    TPR = TP / (TP + FN) if (TP + FN) > 0 else 0
    FPR = FP / (FP + TN) if (FP + TN) > 0 else 0
    TNR = TN / (TN + FP) if (TN + FP) > 0 else 0
    P = TP + FN
    N = TN + FP

    BM = TPR + TNR - 1
    prevalence = P / (P + N) if (P + N) > 0 else np.nan

    FNR = 1 - TPR

    # PPV (Precision) and FDR
    PPV = TP / (TP + FP) if (TP + FP) > 0 else np.nan
    FDR = 1 - PPV if not np.isnan(PPV) else np.nan

    # NPV and FOR
    NPV = TN / (TN + FN) if (TN + FN) > 0 else np.nan
    FOR = 1 - NPV if not np.isnan(NPV) else np.nan

    # Likelihood Ratios and DOR
    LR_plus  = TPR / FPR if FPR > 0 else np.nan
    LR_minus = FNR / TNR if TNR > 0 else np.nan
    DOR = LR_plus / LR_minus if (LR_minus and LR_minus > 0) else np.nan

    ACC = (TP + TN) / (TP + TN + FP + FN) if (TP + TN + FP + FN) > 0 else np.nan
    BA = (TPR + TNR) / 2
    F1 = (2 * PPV * TPR) / (PPV + TPR) if (PPV + TPR) > 0 else np.nan
    FM = np.sqrt(PPV * TPR) if (PPV > 0 and TPR > 0) else 0
    MCC = np.sqrt(TPR * TNR * PPV * NPV) - np.sqrt(FNR * FPR * FOR * FDR)
    TS = TP / (TP + FN + FP) if (TP + FN + FP) > 0 else 0

    results_3.append({
        "Class": col,
```

```
        "BM": round(prevalence, 3),
        "Prev.": round(prevalence, 3),
        "FNR": round(FNR, 3),
        "PPV": round(PPV, 3),
        "FDR": round(FDR, 3),
        "NPV": round(NPV, 3),
        "FOR": round(FOR, 3),
        "LR+": round(LR_plus, 3),
        "LR-": round(LR_minus, 3),
        "DOR": round(DOR, 3),
        "ACC": round(ACC, 3),
        "BA": round(BA, 3),
        "F1": round(F1, 3),
        "FM": round(FM, 3),
        "MCC": round(MCC, 3),
        "TS": round(TS, 3),
    })

matriz_confusao_3 = pd.DataFrame(results_3)
print(matriz_confusao_3.round(3).to_string(index=False))
```

Class	BM	Prev.	FNR	PPV	FDR	NPV	FOR	LR+	LR-	DOR	ACC	BA	F1	FM	MCC	TS
Label1	0.481	0.481	0.0	1.0	0.0	1.0	0.0	NaN	0.0	NaN	1.0	1.0	1.0	1.0	1.0	1.0
Label2	0.011	0.011	0.0	1.0	0.0	1.0	0.0	NaN	0.0	NaN	1.0	1.0	1.0	1.0	1.0	1.0
Label3	0.140	0.140	0.0	1.0	0.0	1.0	0.0	NaN	0.0	NaN	1.0	1.0	1.0	1.0	1.0	1.0
Label4	0.024	0.024	0.0	1.0	0.0	1.0	0.0	NaN	0.0	NaN	1.0	1.0	1.0	1.0	1.0	1.0
Label5	0.022	0.022	0.0	1.0	0.0	1.0	0.0	NaN	0.0	NaN	1.0	1.0	1.0	1.0	1.0	1.0
Label6	0.000	0.000	1.0	NaN	NaN	1.0	0.0	NaN	1.0	NaN	1.0	0.5	NaN	0.0	NaN	0.0
Label7	0.000	0.000	1.0	NaN	NaN	1.0	0.0	NaN	1.0	NaN	1.0	0.5	NaN	0.0	NaN	0.0
Label8	0.032	0.032	0.0	1.0	0.0	1.0	0.0	NaN	0.0	NaN	1.0	1.0	1.0	1.0	1.0	1.0

3. Roc Curves

A ROC curve (Receiver Operating Characteristic) shows the relationship between the TPR (True Positive Rate) = Sensitivity and FPR (False Positive Rate) = 1 − Specificity for different decision thresholds. It is useful for evaluating the overall performance of the model in distinguishing between positives and negatives, regardless of a specific cutoff point.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc, roc_auc_score

# computing for only one class
true_c = true.iloc[:, :1]
pred_c = pred.iloc[:, :1]
fpr, tpr, thresholds = metrics.roc_curve(true_c.values,
```

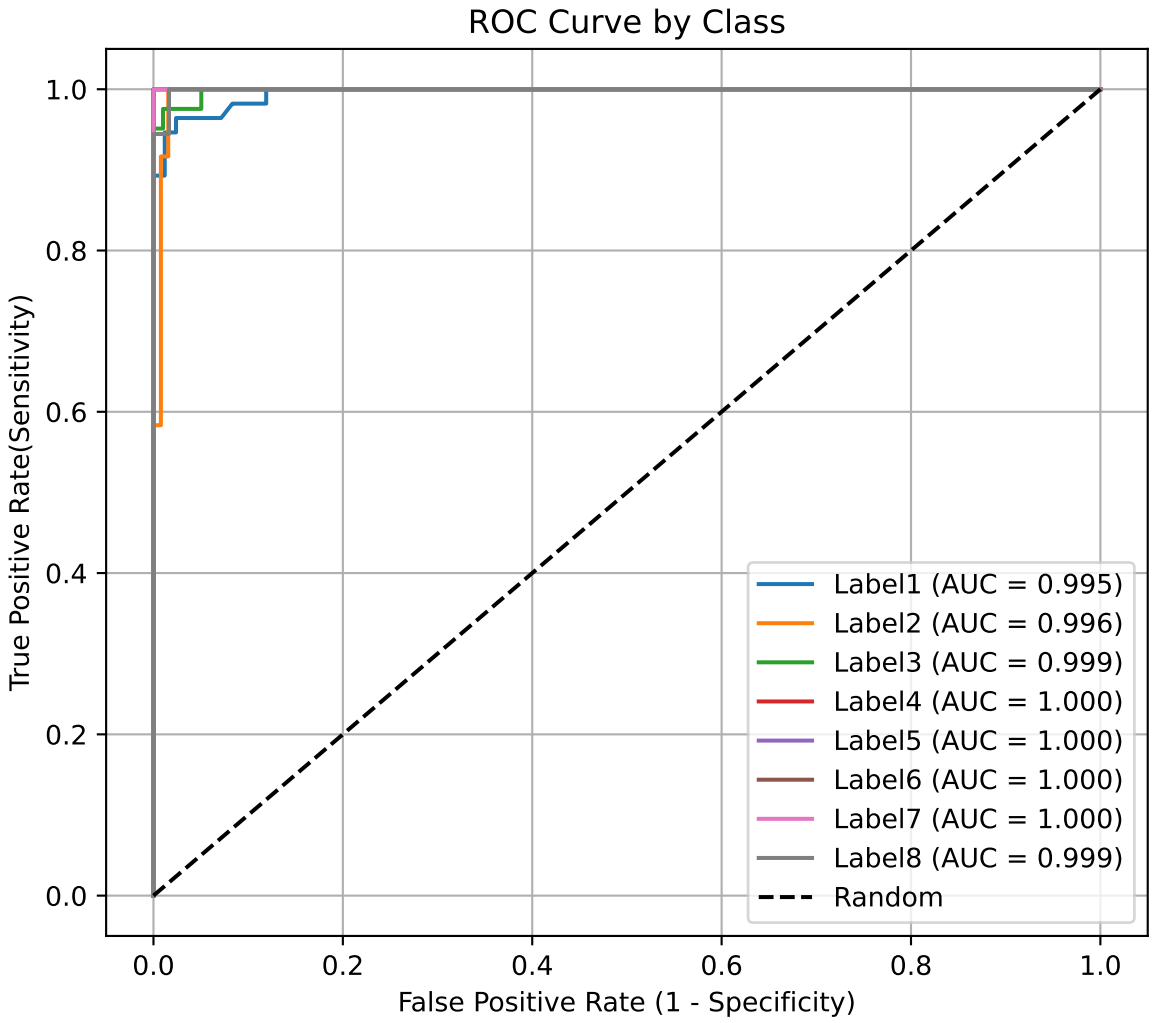
```
                                        pred_c.values)
auc = metrics.auc(fpr, tpr)
print("FPR\n\n", fpr)
print("\n\nTPR\n\n", tpr)
print("\n\nThresholds\n\n", thresholds)
print("\n\nAUC\n\n", auc)
```

```
FPR

 [0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.01190476 0.01190476 0.01190476
 0.02380952 0.02380952 0.07142857 0.08333333 0.10714286 0.11904762
 0.11904762 0.14285714 0.16666667 0.22619048 0.25       0.28571429
 0.29761905 0.33333333 0.35714286 0.36904762 0.4047619  0.42857143
 0.48809524 0.55952381 0.67857143 1.        ]


TPR

 [0.         0.44642857 0.51785714 0.55357143 0.57142857 0.64285714
 0.75       0.85714286 0.89285714 0.89285714 0.92857143 0.94642857
 0.94642857 0.96428571 0.96428571 0.98214286 0.98214286 0.98214286
 1.         1.         1.         1.         1.         1.
 1.         1.         1.         1.         1.         1.
 1.         1.         1.         1.        ]


Thresholds

 [       inf 1.         0.995      0.99       0.98333333 0.98
 0.89666667 0.89       0.81       0.79       0.67       0.645
 0.56958333 0.475      0.24       0.21       0.205      0.135
 0.13333333 0.10083333 0.09091667 0.07       0.06       0.05
 0.04666667 0.045      0.04       0.03625    0.035      0.0175
 0.015      0.01       0.005      0.        ]


AUC

 0.9954294217687075
```

Compute for all Classes!

```
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, roc_auc_score

roc_results = []
```

```
plt.figure(figsize=(7, 6))

for col in true.columns:
    y_true = true[col]
    y_score = pred[col]
    fpr, tpr, thresholds = roc_curve(y_true, y_score)
    auc = roc_auc_score(y_true, y_score)

    # Armazena resultados
    roc_results.append({
        'Class': col,
        'AUC': auc,
    })

    # Plota curva
    plt.plot(fpr, tpr, label=f"{col} (AUC = {auc:.3f})")

# Linha de referência (aleatório)
plt.plot([0, 1], [0, 1], 'k--', label='Random')

plt.xlabel('False Positive Rate (1 - Specificity)')
plt.ylabel('True Positive Rate(Sensitivity)')
plt.title('ROC Curve by Class')
plt.legend()
plt.grid(True)
plt.show()
```



Values:

```
roc_df = pd.DataFrame(roc_results)
print(roc_df)
```

```
    Class       AUC
0  Label1  0.995429
1  Label2  0.996094
2  Label3  0.998522
3  Label4  1.000000
4  Label5  1.000000
5  Label6  1.000000
6  Label7  1.000000
7  Label8  0.999089
```

Now we can compute other types of ROC curves:

- micro: calculate metrics globally by considering each element of the label indicator matrix as a label. Consider all classes at once, weighting them by the number of samples.

- macro: calculate metrics for each label, and find their unweighted mean. This does not take label imbalance into account. Calculates the simple average of the ROC curves for each class.
- weighted: calculate metrics for each label, and find their average, weighted by support (the number of true instances for each label).
- samples: calculate metrics for each instance, and find their average.

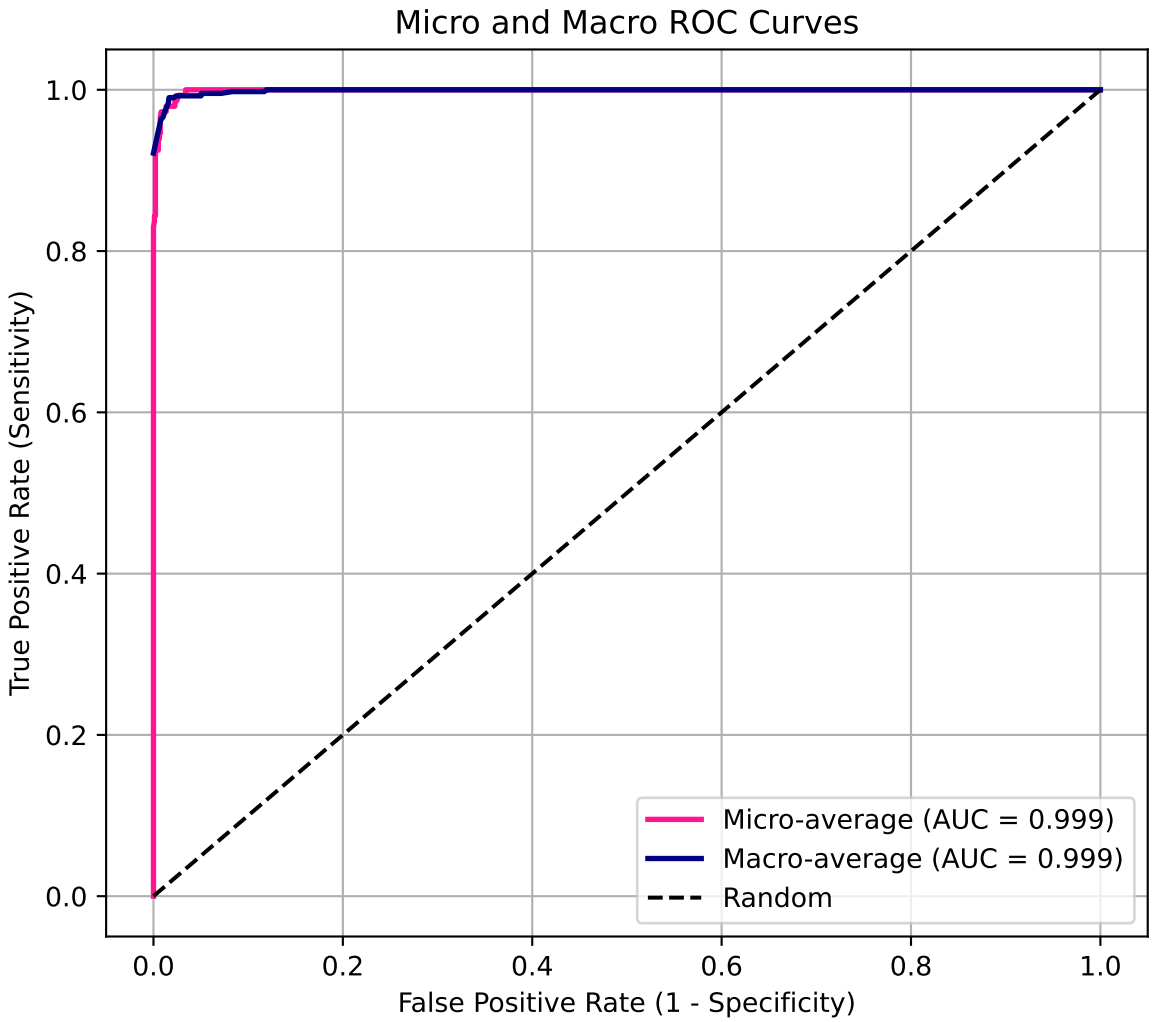
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc, roc_auc_score

roc_auc_macro = roc_auc_score(true, pred, average='macro')
roc_auc_micro = roc_auc_score(true, pred, average='micro')
roc_auc_weighted = roc_auc_score(true, pred, average='weighted')
roc_auc_samples = roc_auc_score(true, pred, average='samples')

print(f"Macro AUC: {roc_auc_macro:.3f}")
print(f"Micro AUC: {roc_auc_micro:.3f}")
print(f"Weighted AUC: {roc_auc_weighted:.3f}")
print(f"Samples AUC: {roc_auc_samples:.3f}")
```

```
Macro AUC: 0.999
Micro AUC: 0.999
Weighted AUC: 0.997
Samples AUC: 0.995
```

And plot the ROC curves:

```
fpr_micro, tpr_micro, _ = roc_curve(true.values.ravel(), pred.values.ravel())

all_fpr = np.unique(np.concatenate([roc_curve(true[col],
pred[col])[0] for col in true.columns]))

mean_tpr = np.zeros_like(all_fpr)

for col in true.columns:
    fpr, tpr, _ = roc_curve(true[col], pred[col])
    mean_tpr += np.interp(all_fpr, fpr, tpr)

mean_tpr /= len(true.columns)

plt.figure(figsize=(7, 6))
plt.plot(fpr_micro, tpr_micro, label=f"Micro-average (AUC = {roc_auc_micro:.3f})",
color="deeppink",
linewidth=2)
plt.plot(all_fpr, mean_tpr,
label=f"Macro-average (AUC = {roc_auc_macro:.3f})",
```

```
color="navy", linewidth=2)
plt.plot([0, 1], [0, 1], 'k--', label='Random')

plt.xlabel("False Positive Rate (1 - Specificity)")
plt.ylabel("True Positive Rate (Sensitivity)")
plt.title("Micro and Macro ROC Curves")
plt.legend()
plt.grid(True)
plt.show()
```



Micro-Averaged in the code: This line flattens (ravels) all the values in your DataFrame – that is, it treats all classes together as if they were a single large vector of binary decisions (a global "one-vs-all"). Interpretation: Each 0/1 value of each label becomes an observation. It's as if you were evaluating all the model's predictions together, without separating them by class. The result is the Micro-average ROC auc curve. It is used to give proportional weight to the number of samples (classes with more positive examples influence the result more).

Macro-Averaged in the code: It collects all possible FPR (False Positive Rate) values for each class and combines them into a single vector (all_fpr). For each class, it interpolates the corresponding TPR (True

Positive Rate). Then, it averages these TPRs across the classes. Interpretation: each class has the same weight (regardless of how many examples it has). The result is the Macro-average ROC curve.

Macro-Averaged Definition

Obtaining the macro-average requires computing the metric independently for each class and then taking the average over them, hence treating all classes equally a priori. We first aggregate the true/false positive rates per class:

$$TPR = \frac{1}{C}\sum_{c}\frac{TP_c}{TP_c + FN_c};$$

$$FPR = \frac{1}{C}\sum_{c}\frac{FP_c}{FP_c + TN_c}.$$

where (C) is the total number of classes.

Micro-Averaged Definition

Obtaining the macro-average requires computing the metric independently for each class and then taking the average over them, hence treating all classes equally a priori. We first aggregate the true/false positive rates per class:

$$TPR = \frac{1}{C}\sum_{c}\frac{TP_c}{TP_c + FN_c};$$

$$FPR = \frac{1}{C}\sum_{c}\frac{FP_c}{FP_c + TN_c}.$$

where (C) is the total number of classes.

3.1. Special Cases

In Multi-Label Classification, some times we have cases where there are no positive or negative values in the true labels. Because of that, the roc curve cannot be compute. But, we can treat that somehow.

3.1.1. True == 1 and Pred == 1

```
true = pd.read_csv("~/AuprcRoc/data-fake/true_1.csv")
pred = pd.read_csv("~/AuprcRoc/data-fake/pred_1.csv")
```

```
print(true)
```

	Label1	Label2	Label3	Label4	Label5
0	1	1	1	1	1
1	1	1	1	1	1
2	1	1	1	1	1
3	1	1	1	1	1
4	1	1	1	1	1
5	1	1	1	1	1
6	1	1	1	1	1
7	1	1	1	1	1
8	1	1	1	1	1
9	1	1	1	1	1

```
print(pred)
```

	Label1	Label2	Label3	Label4	Label5
0	1	1	1	1	1
1	1	1	1	1	1
2	1	1	1	1	1
3	1	1	1	1	1
4	1	1	1	1	1
5	1	1	1	1	1
6	1	1	1	1	1
7	1	1	1	1	1
8	1	1	1	1	1
9	1	1	1	1	1

Lets compute the macro averaged

```
roc_auc_score(true, pred, average='macro')
```

```
/home/cissagatto/miniforge3/envs/ELCC/lib/python3.10/site-packages/sklearn/metrics/_ranking.py:424:
  warnings.warn(
```

```
nan
```

Lets compute the micro averaged

```
roc_auc_score(true, pred, average='micro')
```

```
/home/cissagatto/miniforge3/envs/ELCC/lib/python3.10/site-packages/sklearn/metrics/_ranking.py:424:
  warnings.warn(
```

```
nan
```

Lets compute the weighted averaged

```
roc_auc_score(true, pred, average='weighted')
```

```
/home/cissagatto/miniforge3/envs/ELCC/lib/python3.10/site-packages/sklearn/metrics/_ranking.py:424:
  warnings.warn(
```

```
nan
```

Lets compute the samples averaged

```
roc_auc_score(true, pred, average='samples')
```

```
/home/cissagatto/miniforge3/envs/ELCC/lib/python3.10/site-packages/sklearn/metrics/_ranking.py:424:
  warnings.warn(
```

```
nan
```

So, for every type of ROC curve we receive a NAN as answer.

/home/cissagatto/miniforge3/envs/ELCC/lib/python3.10/ site-packages/sklearn/metrics/__ranking.py:. UndefinedMetricWarning: Only one class is present in y_true. ROC AUC score is not defined in that case. warnings.warn(nan

Why is that? Because we don't have enough positive and negative instances in each class. So, what we can do to, at least, try to have some value? We can determine some rules:

Situation	Assigned AUC Value
`y_true` contains both 0 and 1	Normal AUC
`y_true` contains only 0 and model predicts all 0 (correct)	**1**
`y_true` contains only 0 and model predicts all 1 (wrong)	**0**
`y_true` contains only 1 and model predicts all 1 (correct)	**1**
`y_true` contains only 1 and model predicts all 0 (wrong)	**0**
`y_true` contains only 1 and model predicts both 0 and 1	1
`y_true` contains only 0 and model predicts both 0 and 1	0

Let's try those rules in the following sections. Also, there's a difference in the computation in each type of ROC curve. Let's us cover all of that. First, we compute the ROC curve for only one class to see what is happening.

```
from sklearn import metrics
import pandas as pd

# Example data
true_c = true.iloc[:, :1]
pred_c = pred.iloc[:, :1]
```

```python
# ROC computation
fpr, tpr, thresholds = metrics.roc_curve(true_c.values, pred_c.values)
auc = metrics.auc(fpr, tpr)

# Pretty print
print("=" * 40)
print("ROC Curve Results for Class 1")
print("=" * 40)
print(f"\nAUC: {auc:.4f}")
print("\n--- False Positive Rate (FPR) ---")
print(pd.Series(fpr).to_string(index=False))
print("\n--- True Positive Rate (TPR) ---")
print(pd.Series(tpr).to_string(index=False))
print("\n--- Thresholds ---")
print(pd.Series(thresholds).to_string(index=False))
print("=" * 40)
```

```
========================================
ROC Curve Results for Class 1
========================================

AUC: nan

--- False Positive Rate (FPR) ---
NaN
NaN

--- True Positive Rate (TPR) ---
0.0
1.0

--- Thresholds ---
inf
1.0
========================================
```

```
/home/cissagatto/miniforge3/envs/ELCC/lib/python3.10/site-packages/sklearn/metrics/_ranking.py:1192
  warnings.warn(
```

So, as we can see, we receive NAN in the FPR. As all values of this true and pred as only 1, we will obtain the same message for all labels. For Macro-Average, we can implement our previously mentioned rules! The code below does exactly that.

```python
if len(np.unique(true_c)) < 2:
    if np.all(true_c == 0) and np.all(pred_c == 0):
        fpr = np.array([0, 1])
        tpr = np.array([0, 1])
```

```
        auc = 1
        print(f"{col}: True == 0 and PRED == 0 --> AUC = 1")
    elif np.all(true_c == 1) and np.all(pred_c == 1):
        fpr = np.array([0, 1])
        tpr = np.array([0, 1])
        auc = 1
        print(f"{col}: True == 1 and PRED == 1 --> AUC = 1")
    elif np.all(true_c == 0) and np.all(pred_c == 1):
        fpr = np.array([0, 1])
        tpr = np.array([1, 1])
        auc = 0
        print(f"{col}: True = 0 and PRED = 1 --> AUC = 0")
    elif np.all(true_c == 1) and np.all(pred_c == 0):
        fpr = np.array([0, 1])
        tpr = np.array([0, 0])
        auc = 0
        print(f"{col}: True = 1 and PRED = 0 --> AUC = 0")
    else:
        fpr = np.array([0, 1])
        tpr = np.array([0, 0])
        auc = 0
        print(f"{col}: Other cases")
else:
    fpr, tpr, _ = metrics.roc_curve(true_c, pred_c)
    auc = metrics.auc(fpr, tpr)
```

```
Label8: True == 1 and PRED == 1 --> AUC = 1
```

Ok, now we can extended that for all classes:

```
print("\n" + "="*50)
print("MACRO-AVERAGE ROC")
print("="*50)
fpr_dict_macro = {}
tpr_dict_macro = {}
macro_auc_df = []

for col in true.columns:
    # col = 'Label1'
    y_true = true[col].values
    y_pred = pred[col].values

    if len(np.unique(y_true)) < 2:
        if np.all(y_true == 0) and np.all(y_pred == 0):
            fpr_dict_macro[col] = np.array([0.0, 1.0])
            tpr_dict_macro[col] = np.array([0.0, 1.0])
            auc = 1.0
```

```
            print(f"{col}: True == 0 and PRED == 0 --> AUC = 1")
        elif np.all(y_true == 1) and np.all(y_pred == 1):
            fpr_dict_macro[col] = np.array([0.0, 1.0])
            tpr_dict_macro[col] = np.array([0.0, 1.0])
            auc = 1.0
            print(f"{col}: True == 1 and PRED == 1 --> AUC = 1")
        elif np.all(y_true == 0) and np.all(y_pred == 1):
            fpr_dict_macro[col] = np.array([0.0, 1.0])
            tpr_dict_macro[col] = np.array([1.0, 1.0])
            auc = 0.0
            print(f"{col}: True = 0 and PRED = 1 --> AUC = 0")
        elif np.all(y_true == 1) and np.all(y_pred == 0):
            fpr_dict_macro[col] = np.array([0.0, 1.0])
            tpr_dict_macro[col] = np.array([0.0, 0.0])
            auc = 0.0
            print(f"{col}: True = 1 and PRED = 0 --> AUC = 0")
        else:
            # Caso indefinido
            fpr_dict_macro[col] = np.array([0.0, 1.0])
            tpr_dict_macro[col] = np.array([0.0, 0.0])
            auc = 0.0
            print(f"{col}: Other cases")
    else:
        fpr, tpr, _ = metrics.roc_curve(y_true, y_pred).astype(float)
        fpr_dict_macro[col] = fpr
        tpr_dict_macro[col] = tpr
        print(f"{col}: Normal case")
    macro_auc_df.append(auc)
print("="*50)
```

```
==================================================
MACRO-AVERAGE ROC
==================================================
Label1: True == 1 and PRED == 1 --> AUC = 1
Label2: True == 1 and PRED == 1 --> AUC = 1
Label3: True == 1 and PRED == 1 --> AUC = 1
Label4: True == 1 and PRED == 1 --> AUC = 1
Label5: True == 1 and PRED == 1 --> AUC = 1
==================================================
```

The ROC-AUC result is the average between all classes:

```
macro_auc = np.average(macro_auc_df)
print(macro_auc)
```

```
1.0
```

But, for Micro-Averaged we have to do something different. We can not use the previous code. Now we have to consider the rows not the columns, as following

```
y_true_all = true.values.ravel()
y_pred_all = pred.values.ravel()
```

```
y_true_all
```

```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1])
```

```
y_pred_all
```

```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1])
```

```
np.unique(y_true_all)
```

```
array([1])
```

```
np.unique(y_pred_all)
```

```
array([1])
```

```
len(np.unique(y_true_all))
```

```
1
```

```
len(np.unique(y_true_all))
```

```
1
```

```
# Micro-averaging combines all the pairs and calculates ONE overall ROC curve.
# This gives equal weight to each instance.
print("\n" + "="*40)
print("MICRO-AVERAGE ROC")
print("="*40)

# Detects if only one class is present (all 0s or all 1s).
if len(np.unique(y_true_all)) < 2:
    if np.all(y_true_all == 0) and np.all(y_pred_all == 0):
```

```
        # Defines a "trivial" ROC curve that goes from (0,0) to (1,1)
        # - it's a convention to be able to draw something.
        fpr_micro = np.array([0.0, 1.0])
        tpr_micro = np.array([0.0, 1.0])
        auc_micro = 1.0
        print("True == 0 and PRED == 0 --> AUC = 1")

    elif np.all(y_true_all == 1) and np.all(y_pred_all == 1):
        fpr_micro = np.array([0.0, 1.0])
        tpr_micro = np.array([0.0, 1.0])
        auc_micro = 1.0
        print("True == 1 and PRED == 1 --> AUC = 1")

    elif np.all(y_true_all == 0) and np.all(y_pred_all == 1):
        fpr_micro = np.array([0.0, 1.0])
        tpr_micro = np.array([1.0, 1.0])
        auc_micro = 0.0
        print("True == 0 and PRED == 1 --> AUC = 0")

    elif np.all(y_true_all == 1) and np.all(y_pred_all == 0):
        fpr_micro = np.array([0.0, 1.0])
        tpr_micro = np.array([0.0, 0.0])
        auc_micro = 0.0
        print("True == 1 and PRED == 0 --> AUC = 0")

    else:
        fpr_micro = np.array([0.0, 1.0])
        tpr_micro = np.array([0.0, 0.0])
        auc_micro = 0.0
        print("Other cases --> AUC = 0")

else:
    fpr_micro, tpr_micro, _ = metrics.roc_curve(
        y_true_all.astype(float), y_pred_all.astype(float)
    )
    auc_micro = metrics.auc(fpr_micro, tpr_micro)
    print("normal case.")

print(f"\nAUC micro: {auc_micro:.4f}")
print("="*40)
```

```
========================================
MICRO-AVERAGE ROC
========================================
True == 1 and PRED == 1 --> AUC = 1

AUC micro: 1.0000
```

```
========================================
```

In that way we have the average AUC for macro and micro special cases. We can also compute the interpolated macro averaged:

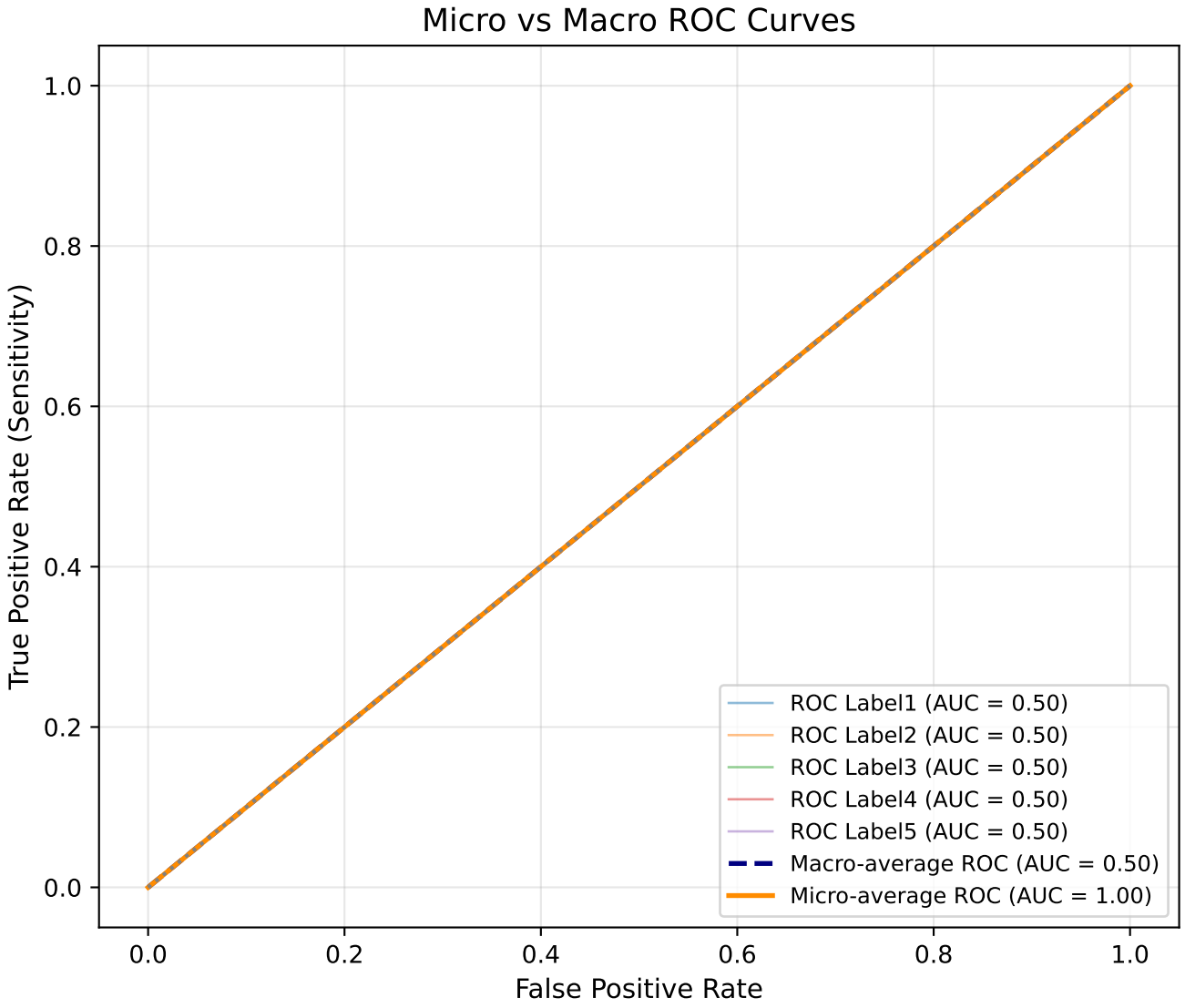
```python
# Interpolated mean ROC curve
all_fpr = np.unique(np.concatenate([fpr_dict_macro[c] for c in fpr_dict_macro]))
mean_tpr = np.zeros_like(all_fpr, dtype=float)
for c in fpr_dict_macro:
    mean_tpr += np.interp(all_fpr, fpr_dict_macro[c], tpr_dict_macro[c])
mean_tpr /= len(fpr_dict_macro)
macro_auc_interp = metrics.auc(all_fpr, mean_tpr)

print(f"Macro AUC interpolated curve: {macro_auc_interp:.4f}")
print(f"Macro AUC interpolated simple: {macro_auc:.4f}")
```

```
Macro AUC interpolated curve: 0.5000
Macro AUC interpolated simple: 1.0000
```

Then we can plot. In the macro-roc-auc you have several curves (one per class), meaning you need to interpolate to average them (since each curve has its own FPR points), while in the micro-roc-auc you only have one overall curve, meaning there is no need to align points, as it is built directly from all the examples together.

```python
# ========================================
# PLOT: MACRO vs MICRO ROC CURVES
# ========================================

import matplotlib.pyplot as plt

plt.figure(figsize=(7, 6))

# Individual Curves
for c in fpr_dict_macro:
    plt.plot(
        fpr_dict_macro[c], tpr_dict_macro[c],
        lw=1, alpha=0.5,
        label=f'ROC {c} (AUC = {metrics.auc(fpr_dict_macro[c], tpr_dict_macro[c]):.2f})'
    )

# Macro-average (interpolated)
plt.plot(
    all_fpr, mean_tpr,
    color='navy', linestyle='--', linewidth=2,
    label=f'Macro-average ROC (AUC = {macro_auc_interp:.2f})'
)

# Micro-average
```

```
plt.plot(
    fpr_micro, tpr_micro,
    color='darkorange', linestyle='-', linewidth=2,
    label=f'Micro-average ROC (AUC = {auc_micro:.2f})'
)

# reference diagonal
plt.plot([0, 1], [0, 1], color='gray', linestyle=':', lw=1.5)

plt.title("Micro vs Macro ROC Curves", fontsize=13)
plt.xlabel("False Positive Rate", fontsize=11)
plt.ylabel("True Positive Rate (Sensitivity)", fontsize=11)
plt.legend(loc="lower right", fontsize=9)
plt.grid(alpha=0.3)
plt.tight_layout()
plt.show()
```



To help us analyises the other cases, I create functions for macro and micro computations, as follows:

```
from sklearn import metrics
import numpy as np

def compute_macro_roc_auc(true, pred, verbose=True):
    """
    Calcula o ROC-AUC macro-average, tratando casos degenerados (sem variação de classe)
    e gerando a curva média interpolada entre classes.

    Parâmetros
    ----------
    true : pandas.DataFrame
        DataFrame com colunas de classes binárias verdadeiras (0/1).
    pred : pandas.DataFrame
        DataFrame com colunas de probabilidades previstas correspondentes.
    verbose : bool, opcional
        Se True, imprime mensagens de status.

    Retorna
    -------
    fpr_dict_macro : dict
        Dicionário com os FPRs por classe.
    tpr_dict_macro : dict
        Dicionário com os TPRs por classe.
    macro_auc : float
        Média simples das AUCs por classe.
    macro_auc_interp : float
        AUC da curva ROC média interpolada (macro-averaged ROC curve).
    """

    fpr_dict_macro = {}
    tpr_dict_macro = {}
    macro_auc_df = []

    if verbose:
        print("\n" + "="*40)
        print("MACRO-AVERAGE ROC")
        print("="*40)

    for col in true.columns:
        y_true = true[col].values
        y_pred = pred[col].values

        if len(np.unique(y_true)) < 2:

            if np.all(y_true == 0) and np.all(y_pred == 0):
                fpr_dict_macro[col] = np.array([0.0, 1.0])
                tpr_dict_macro[col] = np.array([0.0, 1.0])
```

```
            auc = 1.0
            msg = f"{col}: True == 0 and PRED == 0 --> AUC = 1"

        elif np.all(y_true == 1) and np.all(y_pred == 1):
            fpr_dict_macro[col] = np.array([0.0, 1.0])
            tpr_dict_macro[col] = np.array([0.0, 1.0])
            auc = 1.0
            msg = f"{col}: True == 1 and PRED == 1 --> AUC = 1"

        elif np.all(y_true == 0) and np.all(y_pred == 1):
            fpr_dict_macro[col] = np.array([0.0, 1.0])
            tpr_dict_macro[col] = np.array([1.0, 1.0])
            auc = 0.0
            msg = f"{col}: True == 0 and PRED == 1 --> AUC = 0"

        elif np.all(y_true == 1) and np.all(y_pred == 0):
            fpr_dict_macro[col] = np.array([0.0, 1.0])
            tpr_dict_macro[col] = np.array([0.0, 0.0])
            auc = 0.0
            msg = f"{col}: True == 1 and PRED == 0 --> AUC = 0"

        else:
            fpr_dict_macro[col] = np.array([0.0, 1.0])
            tpr_dict_macro[col] = np.array([0.0, 0.0])
            auc = 0.0
            msg = f"{col}: Other cases"

        if verbose:
            print(msg)
    else:
        # Caso normal
        fpr, tpr, _ = metrics.roc_curve(y_true.astype(float), y_pred.astype(float))
        fpr_dict_macro[col] = fpr
        tpr_dict_macro[col] = tpr
        auc = metrics.auc(fpr, tpr)
        if verbose:
            print(f"{col}: Normal case (AUC = {auc:.4f})")

    macro_auc_df.append(auc)

# --- Média simples das AUCs ---
macro_auc = np.mean(macro_auc_df)

# --- Interpolated mean ROC curve ---
all_fpr = np.unique(np.concatenate([fpr_dict_macro[c] for c in fpr_dict_macro]))
mean_tpr = np.zeros_like(all_fpr, dtype=float)

for c in fpr_dict_macro:
```

```
        mean_tpr += np.interp(all_fpr, fpr_dict_macro[c], tpr_dict_macro[c])

    mean_tpr /= len(fpr_dict_macro)
    macro_auc_interp = metrics.auc(all_fpr, mean_tpr)

    if verbose:
        print("\n" + "-"*40)
        print(f"Macro AUC mean of individual AUCs: {macro_auc:.4f}")
        print(f"Macro AUC interpolated mean curve: {macro_auc_interp:.4f}")
        print("="*40)

    return fpr_dict_macro, tpr_dict_macro, macro_auc, macro_auc_interp
```

Using the function

```
fpr_dict_macro, tpr_dict_macro, macro_auc, macro_auc_interp = compute_macro_roc_auc(true, pred)
```

```
========================================
MACRO-AVERAGE ROC
========================================
Label1: True == 1 and PRED == 1 --> AUC = 1
Label2: True == 1 and PRED == 1 --> AUC = 1
Label3: True == 1 and PRED == 1 --> AUC = 1
Label4: True == 1 and PRED == 1 --> AUC = 1
Label5: True == 1 and PRED == 1 --> AUC = 1

----------------------------------------
Macro AUC mean of individual AUCs: 1.0000
Macro AUC interpolated mean curve: 0.5000
========================================
```

Now the micro function

```
from sklearn import metrics
import numpy as np

def compute_micro_roc_auc(true, pred, verbose=True):
    """
    Calcula o ROC-AUC micro-average, tratando casos degenerados (sem variação de classe).

    Parâmetros
    ----------
    true : pandas.DataFrame
        DataFrame com colunas de classes binárias verdadeiras (0/1).
    pred : pandas.DataFrame
        DataFrame com colunas de probabilidades previstas correspondentes.
```

```
    verbose : bool, opcional
        Se True, imprime mensagens de status.

    Retorna
    -------
    fpr_micro : np.ndarray
        False Positive Rate (FPR) da curva micro.
    tpr_micro : np.ndarray
        True Positive Rate (TPR) da curva micro.
    auc_micro : float
        Valor da AUC micro-average.
    """

    # Achata as matrizes (concatena todas as classes)
    y_true_all = true.values.ravel()
    y_pred_all = pred.values.ravel()

    if verbose:
        print("\n" + "="*40)
        print("MICRO-AVERAGE ROC")
        print("="*40)

    # Trata os casos degenerados (sem variação de classe)
    if len(np.unique(y_true_all)) < 2:
        if np.all(y_true_all == 0) and np.all(y_pred_all == 0):
            fpr_micro = np.array([0.0, 1.0])
            tpr_micro = np.array([0.0, 1.0])
            auc_micro = 1.0
            msg = "True == 0 and PRED == 0 --> AUC = 1"
        elif np.all(y_true_all == 1) and np.all(y_pred_all == 1):
            fpr_micro = np.array([0.0, 1.0])
            tpr_micro = np.array([0.0, 1.0])
            auc_micro = 1.0
            msg = "True == 1 and PRED == 1 --> AUC = 1"
        elif np.all(y_true_all == 0) and np.all(y_pred_all == 1):
            fpr_micro = np.array([0.0, 1.0])
            tpr_micro = np.array([1.0, 1.0])
            auc_micro = 0.0
            msg = "True == 0 and PRED == 1 --> AUC = 0"
        elif np.all(y_true_all == 1) and np.all(y_pred_all == 0):
            fpr_micro = np.array([0.0, 1.0])
            tpr_micro = np.array([0.0, 0.0])
            auc_micro = 0.0
            msg = "True == 1 and PRED == 0 --> AUC = 0"
        else:
            fpr_micro = np.array([0.0, 1.0])
            tpr_micro = np.array([0.0, 0.0])
            auc_micro = 0.0
```

```
            msg = "Other cases --> AUC = 0"

        if verbose:
            print(msg)

    else:
        # Caso normal - calcula ROC e AUC reais
        fpr_micro, tpr_micro, _ = metrics.roc_curve(
            y_true_all.astype(float), y_pred_all.astype(float)
        )
        auc_micro = metrics.auc(fpr_micro, tpr_micro)
        if verbose:
            print("Normal case.")

    if verbose:
        print(f"\nAUC micro: {auc_micro:.4f}")
        print("="*40)

    return fpr_micro, tpr_micro, auc_micro
```

Now using the function:

```
fpr_micro, tpr_micro, auc_micro = compute_micro_roc_auc(true, pred)
```

```
========================================
MICRO-AVERAGE ROC
========================================
True == 1 and PRED == 1 --> AUC = 1

AUC micro: 1.0000
========================================
```

3.1.2. True == 0 and Pred == 0

```
true = pd.read_csv("~/AuprcRoc/data-fake/true_0.csv")
pred = pd.read_csv("~/AuprcRoc/data-fake/pred_0.csv")
```

Macro-Average

```
fpr_dict_macro, tpr_dict_macro, macro_auc, macro_auc_interp = compute_macro_roc_auc(true, pred)
```

```
========================================
MACRO-AVERAGE ROC
========================================
```

```
Label1: True == 0 and PRED == 0 --> AUC = 1
Label2: True == 0 and PRED == 0 --> AUC = 1
Label3: True == 0 and PRED == 0 --> AUC = 1
Label4: True == 0 and PRED == 0 --> AUC = 1
Label5: True == 0 and PRED == 0 --> AUC = 1

----------------------------------------
Macro AUC mean of individual AUCs: 1.0000
Macro AUC interpolated mean curve: 0.5000
========================================
```

Micro-Averaged

```
fpr_micro, tpr_micro, auc_micro = compute_micro_roc_auc(true, pred)
```

```
========================================
MICRO-AVERAGE ROC
========================================
True == 0 and PRED == 0 --> AUC = 1

AUC micro: 1.0000
========================================
```

3.1.3. True == 0 and Pred == 1

```
true = pd.read_csv("~/AuprcRoc/data-fake/true_0.csv")
pred = pd.read_csv("~/AuprcRoc/data-fake/pred_1.csv")
```

Macro-Average

```
fpr_dict_macro, tpr_dict_macro, macro_auc, macro_auc_interp = compute_macro_roc_auc(true, pred)
```

```
========================================
MACRO-AVERAGE ROC
========================================
Label1: True == 0 and PRED == 1 --> AUC = 0
Label2: True == 0 and PRED == 1 --> AUC = 0
Label3: True == 0 and PRED == 1 --> AUC = 0
Label4: True == 0 and PRED == 1 --> AUC = 0
Label5: True == 0 and PRED == 1 --> AUC = 0

----------------------------------------
Macro AUC mean of individual AUCs: 0.0000
Macro AUC interpolated mean curve: 1.0000
========================================
```

Micro-Averaged

```
fpr_micro, tpr_micro, auc_micro = compute_micro_roc_auc(true, pred)
```

```
========================================
MICRO-AVERAGE ROC
========================================
True == 0 and PRED == 1 --> AUC = 0

AUC micro: 0.0000
========================================
```

3.1.4. True == 1 and Pred == 0

```
true = pd.read_csv("~/AuprcRoc/data-fake/true_1.csv")
pred = pd.read_csv("~/AuprcRoc/data-fake/pred_0.csv")
```

Macro-Average

```
fpr_dict_macro, tpr_dict_macro, macro_auc, macro_auc_interp = compute_macro_roc_auc(true, pred)
```

```
========================================
MACRO-AVERAGE ROC
========================================
Label1: True == 1 and PRED == 0 --> AUC = 0
Label2: True == 1 and PRED == 0 --> AUC = 0
Label3: True == 1 and PRED == 0 --> AUC = 0
Label4: True == 1 and PRED == 0 --> AUC = 0
Label5: True == 1 and PRED == 0 --> AUC = 0

----------------------------------------
Macro AUC mean of individual AUCs: 0.0000
Macro AUC interpolated mean curve: 0.0000
========================================
```

Micro-Averaged

```
fpr_micro, tpr_micro, auc_micro = compute_micro_roc_auc(true, pred)
```

```
========================================
MICRO-AVERAGE ROC
========================================
```

```
True == 1 and PRED == 0 --> AUC = 0

AUC micro: 0.0000
========================================
```

3.1.5. True == 1 and Pred == 0's and 1's

```
true = pd.read_csv("~/AuprcRoc/data-fake/true_1.csv")
pred = pd.read_csv("~/AuprcRoc/data-fake/pred_3.csv")
```

Macro-Average

```
fpr_dict_macro, tpr_dict_macro, macro_auc, macro_auc_interp = compute_macro_roc_auc(true, pred)
```

```
========================================
MACRO-AVERAGE ROC
========================================
Label1: Other cases
Label2: Other cases
Label3: Other cases
Label4: Other cases
Label5: Other cases

----------------------------------------
Macro AUC mean of individual AUCs: 0.0000
Macro AUC interpolated mean curve: 0.0000
========================================
```

Micro-Averaged

```
fpr_micro, tpr_micro, auc_micro = compute_micro_roc_auc(true, pred)
```

```
========================================
MICRO-AVERAGE ROC
========================================
Other cases --> AUC = 0

AUC micro: 0.0000
========================================
```

```
true = pd.read_csv("~/AuprcRoc/data-fake/true_1.csv")
pred = pd.read_csv("~/AuprcRoc/data-fake/pred_3.csv")
```

Macro-Average

```
fpr_dict_macro, tpr_dict_macro, macro_auc, macro_auc_interp = compute_macro_roc_auc(true, pred)
```

```
========================================
MACRO-AVERAGE ROC
========================================
Label1: Other cases
Label2: Other cases
Label3: Other cases
Label4: Other cases
Label5: Other cases

----------------------------------------
Macro AUC mean of individual AUCs: 0.0000
Macro AUC interpolated mean curve: 0.0000
========================================
```

Micro-Averaged

```
fpr_micro, tpr_micro, auc_micro = compute_micro_roc_auc(true, pred)
```

```
========================================
MICRO-AVERAGE ROC
========================================
Other cases --> AUC = 0

AUC micro: 0.0000
========================================
```

```
true = pd.read_csv("~/AuprcRoc/data-fake/true_1.csv")
pred = pd.read_csv("~/AuprcRoc/data-fake/pred_2.csv")
```

Macro-Average

```
fpr_dict_macro, tpr_dict_macro, macro_auc, macro_auc_interp = compute_macro_roc_auc(true, pred)
```

```
========================================
MACRO-AVERAGE ROC
========================================
Label1: True == 1 and PRED == 1 --> AUC = 1
Label2: True == 1 and PRED == 0 --> AUC = 0
Label3: True == 1 and PRED == 1 --> AUC = 1
Label4: True == 1 and PRED == 0 --> AUC = 0
```

```
Label5: True == 1 and PRED == 1 --> AUC = 1

----------------------------------------
Macro AUC mean of individual AUCs: 0.6000
Macro AUC interpolated mean curve: 0.3000
========================================
```

Micro-Averaged

```
fpr_micro, tpr_micro, auc_micro = compute_micro_roc_auc(true, pred)
```

```
========================================
MICRO-AVERAGE ROC
========================================
Other cases --> AUC = 0

AUC micro: 0.0000
========================================
```

3.1.6. True == 0 and Pred == 0's and 1's

```
true = pd.read_csv("~/AuprcRoc/data-fake/true_0.csv")
pred = pd.read_csv("~/AuprcRoc/data-fake/pred_3.csv")
```

Macro-Average

```
fpr_dict_macro, tpr_dict_macro, macro_auc, macro_auc_interp = compute_macro_roc_auc(true, pred)
```

```
========================================
MACRO-AVERAGE ROC
========================================
Label1: Other cases
Label2: Other cases
Label3: Other cases
Label4: Other cases
Label5: Other cases

----------------------------------------
Macro AUC mean of individual AUCs: 0.0000
Macro AUC interpolated mean curve: 0.0000
========================================
```

Micro-Averaged

```
fpr_micro, tpr_micro, auc_micro = compute_micro_roc_auc(true, pred)
```

```
========================================
MICRO-AVERAGE ROC
========================================
Other cases --> AUC = 0

AUC micro: 0.0000
========================================
```

```
true = pd.read_csv("~/AuprcRoc/data-fake/true_0.csv")
pred = pd.read_csv("~/AuprcRoc/data-fake/pred_3.csv")
```

Macro-Average

```
fpr_dict_macro, tpr_dict_macro, macro_auc, macro_auc_interp = compute_macro_roc_auc(true, pred)
```

```
========================================
MACRO-AVERAGE ROC
========================================
Label1: Other cases
Label2: Other cases
Label3: Other cases
Label4: Other cases
Label5: Other cases

----------------------------------------
Macro AUC mean of individual AUCs: 0.0000
Macro AUC interpolated mean curve: 0.0000
========================================
```

Micro-Averaged

```
fpr_micro, tpr_micro, auc_micro = compute_micro_roc_auc(true, pred)
```

```
========================================
MICRO-AVERAGE ROC
========================================
Other cases --> AUC = 0

AUC micro: 0.0000
========================================
```

Macro-Average

```
fpr_dict_macro, tpr_dict_macro, macro_auc, macro_auc_interp = compute_macro_roc_auc(true, pred)
```

```
========================================
MACRO-AVERAGE ROC
========================================
Label1: Other cases
Label2: Other cases
Label3: Other cases
Label4: Other cases
Label5: Other cases

----------------------------------------
Macro AUC mean of individual AUCs: 0.0000
Macro AUC interpolated mean curve: 0.0000
========================================
```

```
true = pd.read_csv("~/AuprcRoc/data-fake/true_0.csv")
pred = pd.read_csv("~/AuprcRoc/data-fake/pred_2.csv")
```

```
true
```

	Label1	Label2	Label3	Label4	Label5
0	0	0	0	0	0
1	0	0	0	0	0
2	0	0	0	0	0
3	0	0	0	0	0
4	0	0	0	0	0
5	0	0	0	0	0
6	0	0	0	0	0
7	0	0	0	0	0
8	0	0	0	0	0
9	0	0	0	0	0

```
pred
```

	Label1	Label2	Label3	Label4	Label5
0	1	0	1	0	1
1	1	0	1	0	1
2	1	0	1	0	1
3	1	0	1	0	1
4	1	0	1	0	1

	Label1	Label2	Label3	Label4	Label5
5	1	0	1	0	1
6	1	0	1	0	1
7	1	0	1	0	1
8	1	0	1	0	1
9	1	0	1	0	1

Macro-Average

```
fpr_dict_macro, tpr_dict_macro, macro_auc, macro_auc_interp = compute_macro_roc_auc(true, pred)
```

```
========================================
MACRO-AVERAGE ROC
========================================
Label1: True == 0 and PRED == 1 --> AUC = 0
Label2: True == 0 and PRED == 0 --> AUC = 1
Label3: True == 0 and PRED == 1 --> AUC = 0
Label4: True == 0 and PRED == 0 --> AUC = 1
Label5: True == 0 and PRED == 1 --> AUC = 0

----------------------------------------
Macro AUC mean of individual AUCs: 0.4000
Macro AUC interpolated mean curve: 0.8000
========================================
```

Micro-Averaged

```
fpr_micro, tpr_micro, auc_micro = compute_micro_roc_auc(true, pred)
```

```
========================================
MICRO-AVERAGE ROC
========================================
Other cases --> AUC = 0

AUC micro: 0.0000
========================================
```

3.2. Detailing the different averages: Macro, Micro, Weighted and Samples

Macro-average

- Calculates the AUC of each class separately (1 vs. rest).
- Then takes a simple average between them.

- Each class has the same weight, regardless of how many positive samples it has.
- Good when: you want to evaluate all classes equally, even the rare ones.
- Example: Class A (100 samples), Class B (10), Class C (1), all count equally in the average.

Weighted-average

- Calculates the AUC of each class separately, but each value is weighted by the number of positive samples of that class.
- More frequent classes have more influence on the final result.
- Good when: there is class imbalance and you want to reflect that in the metric.
- Example: Class A dominates the result if it has many more positive instances than the others.

Micro-average

- Combines all classes and samples into a single flat vector (*flatten*).
- Calculates the AUC globally, treating each pair (sample, class) as an observation.
- Measures the overall performance of the model, without distinguishing classes or samples.
- Good when: you want an overview of the total performance.
- Imagine 3 classes and 5 samples, resulting in 15 decisions (5×3). The micro-average transforms all these decisions into a single set of pairs (y_true, y_pred) and calculates a global ROC curve, considering all points together. Thus, more frequent classes naturally have more weight, as they generate more pairs (positive and negative).

Samples-average

- Calculates the AUC individually per sample (row), considering its labels as a mini binary problem.
- Then takes a simple average across all samples.
- Each sample has the same weight, regardless of the number of positive labels.
- Good when: you want to evaluate the model from the perspective of each instance.
- Example: Each row of the dataset is evaluated individually and contributes equally to the result.

Summary

Type	Weighting	Focus	Useful for
Macro	Simple average	Each class counts equally	Evaluating balance between classes
Weighted	Weighted by the number of positives (support)	Frequent classes	Imbalanced data
Micro	Global (instance × label)	All pairs (y_true, y_pred)	Overview
Samples	Average per instance (row)	Each sample counts equally	Evaluate example by example

3.3. Final Roc Auc Macro

As we can see, we have a lot of OTHER CASES, and that is something that shouldnt happen. So here is a new version of the computation that include Mulan rules:

We treat special cases with empty set predictions or/and ground truth as follows:

(i) if the algorithm outputs the empty set and the ground truth is the empty set, then we consider F equal to 1

(ii) if the algorithm outputs the empty set and the ground truth is not empty, then we consider F equal to 0

(iii) if the ground truth is empty and the algorithm does not output the empty set, then we consider F equal to 0

(iv) if neither the ground truth nor the algorithm's prediction is the empty set and their intersection is empty, then we consider F equal to 0.

```
from sklearn import metrics
import numpy as np
import pandas as pd

def robust_macro_roc_auc(true, pred, verbose=True):
    """
    Compute a robust macro-average ROC-AUC with interpolation and special cases handling.

    ------------------------------------------------------------------------
    Description
    ------------------------------------------------------------------------
    This function computes the macro-average ROC-AUC across multiple binary
    labels (multi-label classification). It is designed to handle special
    cases (e.g., when a label has only one class or when predictions are all (0/1)

    ------------------------------------------------------------------------
    Objective
    ------------------------------------------------------------------------
    To provide a stable and interpretable computation of macro-averaged ROC-AUC
    scores even when certain labels have no class variation or when ROC curves
    cannot be defined by standard metrics.

    ------------------------------------------------------------------------
    Parameters
    ------------------------------------------------------------------------
    true : pandas.DataFrame
        Ground truth binary indicators (0 or 1) for each label column.

    pred : pandas.DataFrame
        Predicted scores or probabilities for each label column.
        Can contain either binary values (0/1) or continuous probabilities [0, 1].

    verbose : bool, default=True
        If True, prints detailed information for each label and the macro results.

    ------------------------------------------------------------------------
    Returns
    ------------------------------------------------------------------------
    fpr_dict_macro : dict
```

```
    A dictionary mapping each label to its computed False Positive Rates.

tpr_dict_macro : dict
    A dictionary mapping each label to its computed True Positive Rates.

macro_auc : float
    The unweighted mean of individual label AUCs.

macro_auc_interp : float
    The interpolated macro-average AUC calculated from the mean ROC curve.

macro_auc_df : pandas.DataFrame
    A DataFrame containing one row per label:
        - "Label": label name
        - "AUC": computed AUC value

------------------------------------------------------------------------
Example
------------------------------------------------------------------------
>>> import pandas as pd
>>> from sklearn.datasets import make_multilabel_classification
>>> from sklearn.linear_model import LogisticRegression

>>> X, Y = make_multilabel_classification(n_samples=100, n_features=10, n_classes=3)
>>> true = pd.DataFrame(Y, columns=["Label1", "Label2", "Label3"])

>>> model = LogisticRegression()
>>> pred = pd.DataFrame({
...     "Label1": np.random.rand(100),
...     "Label2": np.random.rand(100),
...     "Label3": np.random.rand(100)
... })

>>> fpr, tpr, macro_auc, macro_auc_interp, macro_auc_df = robust_macro_roc_auc(true, pred)
>>> print(macro_auc_df)
       Label    AUC
0     Label1  0.632
1     Label2  0.701
2     Label3  0.648
"""

fpr_dict_macro = {}
tpr_dict_macro = {}
macro_auc_df = []

if verbose:
    print("\n" + "="*40)
    print("MACRO-AVERAGE ROC")
```

```
        print("="*40)

    # --- Iterate over each label (column) ---
    for col in true.columns:
        y_true = true[col].values
        y_pred = pred[col].values

        # --- CASE 1: y_true has only one class (no variation) ---
        if len(np.unique(y_true)) < 2:

            # (i) both empty (True=0, Pred=0)
            if np.all(y_true == 0) and np.all(y_pred == 0):
                auc = 1.0
                fpr_dict_macro[col] = np.array([0.0, 1.0])
                tpr_dict_macro[col] = np.array([0.0, 1.0])
                msg = f"{col}: True == 0 and PRED == 0 --> AUC = 1"

            # (ii) both full (True=1, Pred=1)
            elif np.all(y_true == 1) and np.all(y_pred == 1):
                auc = 1.0
                fpr_dict_macro[col] = np.array([0.0, 1.0])
                tpr_dict_macro[col] = np.array([0.0, 1.0])
                msg = f"{col}: True == 1 and PRED == 1 --> AUC = 1"

            # (iii) True empty, Pred full
            elif np.all(y_true == 0) and np.all(y_pred == 1):
                auc = 0.0
                fpr_dict_macro[col] = np.array([0.0, 1.0])
                tpr_dict_macro[col] = np.array([1.0, 1.0])
                msg = f"{col}: True == 0 and PRED == 1 --> AUC = 0"

            # (iv) True full, Pred empty
            elif np.all(y_true == 1) and np.all(y_pred == 0):
                auc = 0.0
                fpr_dict_macro[col] = np.array([0.0, 1.0])
                tpr_dict_macro[col] = np.array([0.0, 0.0])
                msg = f"{col}: True == 1 and PRED == 0 --> AUC = 0"

            # (v) True constant (0 or 1), but PRED has probabilities (not 0/1)
            elif len(np.unique(y_true)) == 1 and not np.all(np.isin(y_pred, [0, 1])):
                auc = 0.5
                fpr_dict_macro[col] = np.array([0.0, 1.0])
                tpr_dict_macro[col] = np.array([0.0, 1.0])
                true_val = int(y_true[0])  # vai ser 0 ou 1
                msg = f"{col}: True == {true_val} and PRED probabilistic --> AUC = 0.5"


            # (vi) True all 1, Pred mixed (0/1)
```

```
        elif np.all(y_true == 1) and np.any(y_pred == 0) and np.any(y_pred == 1):
            auc = 1.0
            fpr_dict_macro[col] = np.array([0.0, 1.0])
            tpr_dict_macro[col] = np.array([0.0, 1.0])
            msg = f"{col}: True == 1 but PRED == 0/1 --> AUC = 1"

        # (vii) True all 0, Pred mixed (0/1)
        elif np.all(y_true == 0) and np.any(y_pred == 0) and np.any(y_pred == 1):
            auc = 0.0
            fpr_dict_macro[col] = np.array([0.0, 1.0])
            tpr_dict_macro[col] = np.array([0.0, 0.0])
            msg = f"{col}: True == 0 but PRED == 0/1 --> AUC = 0"

        # fallback (shouldn't happen)
        else:
            auc = 0.5
            fpr_dict_macro[col] = np.array([0.0, 1.0])
            tpr_dict_macro[col] = np.array([0.0, 1.0])
            msg = f"{col}: Other special case --> AUC = 0.5"

        if verbose:
            print(msg)

    # --- CASE 2: normal ROC ---
    else:
        try:
            fpr, tpr, _ = metrics.roc_curve(y_true.astype(float), y_pred.astype(float))
            auc = metrics.auc(fpr, tpr)
            fpr_dict_macro[col] = fpr
            tpr_dict_macro[col] = tpr
            if verbose:
                print(f"{col}: Normal case (AUC = {auc:.4f})")
        except ValueError:
            fpr_dict_macro[col] = np.array([0.0, 1.0])
            tpr_dict_macro[col] = np.array([0.0, 0.0])
            auc = 0.0
            if verbose:
                print(f"{col}: ROC computation failed --> AUC = 0")

    macro_auc_df.append((col, auc))

# --- Convert AUC results to DataFrame ---
macro_auc_df = pd.DataFrame(
    [(label,
      auc,
      fpr_dict_macro[label],
      tpr_dict_macro[label])
     for label, auc in macro_auc_df],
```

```
        columns=["Label", "AUC", "FPR", "TPR"]
    )
    # --- Mean of AUCs ---
    macro_auc = np.mean(macro_auc_df["AUC"])

    # --- Interpolated mean ROC ---
    all_fpr = np.unique(np.concatenate([fpr_dict_macro[c] for c in fpr_dict_macro]))
    mean_tpr = np.zeros_like(all_fpr, dtype=float)

    for c in fpr_dict_macro:
        mean_tpr += np.interp(all_fpr, fpr_dict_macro[c], tpr_dict_macro[c])

    mean_tpr /= len(fpr_dict_macro)
    macro_auc_interp = metrics.auc(all_fpr, mean_tpr)

    if verbose:
        print("\n" + "-"*40)
        print(f"Macro AUC mean of individual AUCs: {macro_auc:.4f}")
        print(f"Macro AUC interpolated mean curve: {macro_auc_interp:.4f}")
        print("="*40)

    return fpr_dict_macro, tpr_dict_macro, macro_auc, macro_auc_interp, macro_auc_df
```

Testing the new function with TRUE == 0 AND PRED == 0

```
true = pd.read_csv("~/AuprcRoc/data-fake/true_0.csv")
pred = pd.read_csv("~/AuprcRoc/data-fake/pred_0.csv")
fpr_dict, tpr_dict, macro_auc, macro_auc_interp, macro_auc_df = robust_macro_roc_auc(true, pred)
```

```
========================================
MACRO-AVERAGE ROC
========================================
Label1: True == 0 and PRED == 0 --> AUC = 1
Label2: True == 0 and PRED == 0 --> AUC = 1
Label3: True == 0 and PRED == 0 --> AUC = 1
Label4: True == 0 and PRED == 0 --> AUC = 1
Label5: True == 0 and PRED == 0 --> AUC = 1

----------------------------------------
Macro AUC mean of individual AUCs: 1.0000
Macro AUC interpolated mean curve: 0.5000
========================================
```

```
fpr_dict
```

```
{'Label1': array([0., 1.]),
```

```
 'Label2': array([0., 1.]),
 'Label3': array([0., 1.]),
 'Label4': array([0., 1.]),
 'Label5': array([0., 1.])}
```

```
tpr_dict
```

```
{'Label1': array([0., 1.]),
 'Label2': array([0., 1.]),
 'Label3': array([0., 1.]),
 'Label4': array([0., 1.]),
 'Label5': array([0., 1.])}
```

```
macro_auc
```

```
np.float64(1.0)
```

```
macro_auc_interp
```

```
0.5
```

```
macro_auc_df
```

	Label	AUC	FPR	TPR
0	Label1	1.0	[0.0, 1.0]	[0.0, 1.0]
1	Label2	1.0	[0.0, 1.0]	[0.0, 1.0]
2	Label3	1.0	[0.0, 1.0]	[0.0, 1.0]
3	Label4	1.0	[0.0, 1.0]	[0.0, 1.0]
4	Label5	1.0	[0.0, 1.0]	[0.0, 1.0]

Testing the new function with TRUE == 0 AND PRED == 1

```
true = pd.read_csv("~/AuprcRoc/data-fake/true_0.csv")
pred = pd.read_csv("~/AuprcRoc/data-fake/pred_1.csv")
fpr_dict, tpr_dict, macro_auc, macro_auc_interp, macro_auc_df = robust_macro_roc_auc(true, pred)
```

```
========================================
MACRO-AVERAGE ROC
========================================
Label1: True == 0 and PRED == 1 --> AUC = 0
Label2: True == 0 and PRED == 1 --> AUC = 0
Label3: True == 0 and PRED == 1 --> AUC = 0
Label4: True == 0 and PRED == 1 --> AUC = 0
```

```
Label5: True == 0 and PRED == 1 --> AUC = 0

----------------------------------------
Macro AUC mean of individual AUCs: 0.0000
Macro AUC interpolated mean curve: 1.0000
========================================
```

Testing the new function with TRUE == 0 AND PRED == 1/0s

```
true = pd.read_csv("~/AuprcRoc/data-fake/true_0.csv")
pred = pd.read_csv("~/AuprcRoc/data-fake/pred_2.csv")
fpr_dict, tpr_dict, macro_auc, macro_auc_interp, macro_auc_df = robust_macro_roc_auc(true, pred)
```

```
========================================
MACRO-AVERAGE ROC
========================================
Label1: True == 0 and PRED == 1 --> AUC = 0
Label2: True == 0 and PRED == 0 --> AUC = 1
Label3: True == 0 and PRED == 1 --> AUC = 0
Label4: True == 0 and PRED == 0 --> AUC = 1
Label5: True == 0 and PRED == 1 --> AUC = 0

----------------------------------------
Macro AUC mean of individual AUCs: 0.4000
Macro AUC interpolated mean curve: 0.8000
========================================
```

Testing the new function with TRUE == 0 AND PRED == 1/0s

```
true = pd.read_csv("~/AuprcRoc/data-fake/true_0.csv")
pred = pd.read_csv("~/AuprcRoc/data-fake/pred_3.csv")
fpr_dict, tpr_dict, macro_auc, macro_auc_interp, macro_auc_df = robust_macro_roc_auc(true, pred)
```

```
========================================
MACRO-AVERAGE ROC
========================================
Label1: True == 0 but PRED == 0/1 --> AUC = 0
Label2: True == 0 but PRED == 0/1 --> AUC = 0
Label3: True == 0 but PRED == 0/1 --> AUC = 0
Label4: True == 0 but PRED == 0/1 --> AUC = 0
Label5: True == 0 but PRED == 0/1 --> AUC = 0

----------------------------------------
Macro AUC mean of individual AUCs: 0.0000
Macro AUC interpolated mean curve: 0.0000
========================================
```

Testing the new function with TRUE == 0 AND PRED == 1/0s

```
true = pd.read_csv("~/AuprcRoc/data-fake/true_0.csv")
pred = pd.read_csv("~/AuprcRoc/data-fake/pred_4.csv")
fpr_dict, tpr_dict, macro_auc, macro_auc_interp, macro_auc_df = robust_macro_roc_auc(true, pred)
```

```
========================================
MACRO-AVERAGE ROC
========================================
Label1: True == 0 but PRED == 0/1 --> AUC = 0
Label2: True == 0 but PRED == 0/1 --> AUC = 0
Label3: True == 0 but PRED == 0/1 --> AUC = 0
Label4: True == 0 but PRED == 0/1 --> AUC = 0
Label5: True == 0 but PRED == 0/1 --> AUC = 0

----------------------------------------
Macro AUC mean of individual AUCs: 0.0000
Macro AUC interpolated mean curve: 0.0000
========================================
```

Testing the new function with TRUE == 0 AND PRED == probabilities

```
true = pd.read_csv("~/AuprcRoc/data-fake/true_0.csv")
pred = pd.read_csv("~/AuprcRoc/data-fake/pred_5.csv")
fpr_dict, tpr_dict, macro_auc, macro_auc_interp, macro_auc_df =robust_macro_roc_auc(true, pred)
```

```
========================================
MACRO-AVERAGE ROC
========================================
Label1: True == 0 and PRED probabilistic --> AUC = 0.5
Label2: True == 0 and PRED probabilistic --> AUC = 0.5
Label3: True == 0 and PRED probabilistic --> AUC = 0.5
Label4: True == 0 and PRED probabilistic --> AUC = 0.5
Label5: True == 0 and PRED probabilistic --> AUC = 0.5

----------------------------------------
Macro AUC mean of individual AUCs: 0.5000
Macro AUC interpolated mean curve: 0.5000
========================================
```

Testing the new function with TRUE == 1 AND PRED == 1

```
true = pd.read_csv("~/AuprcRoc/data-fake/true_1.csv")
pred = pd.read_csv("~/AuprcRoc/data-fake/pred_1.csv")
fpr_dict, tpr_dict, macro_auc, macro_auc_interp, macro_auc_df = robust_macro_roc_auc(true, pred)
```

```
========================================
MACRO-AVERAGE ROC
========================================
Label1: True == 1 and PRED == 1 --> AUC = 1
Label2: True == 1 and PRED == 1 --> AUC = 1
Label3: True == 1 and PRED == 1 --> AUC = 1
Label4: True == 1 and PRED == 1 --> AUC = 1
Label5: True == 1 and PRED == 1 --> AUC = 1

----------------------------------------
Macro AUC mean of individual AUCs: 1.0000
Macro AUC interpolated mean curve: 0.5000
========================================
```

Testing the new function with TRUE == 1 AND PRED == 0

```
true = pd.read_csv("~/AuprcRoc/data-fake/true_1.csv")
pred = pd.read_csv("~/AuprcRoc/data-fake/pred_0.csv")
fpr_dict, tpr_dict, macro_auc, macro_auc_interp, macro_auc_df = robust_macro_roc_auc(true, pred)
```

```
========================================
MACRO-AVERAGE ROC
========================================
Label1: True == 1 and PRED == 0 --> AUC = 0
Label2: True == 1 and PRED == 0 --> AUC = 0
Label3: True == 1 and PRED == 0 --> AUC = 0
Label4: True == 1 and PRED == 0 --> AUC = 0
Label5: True == 1 and PRED == 0 --> AUC = 0

----------------------------------------
Macro AUC mean of individual AUCs: 0.0000
Macro AUC interpolated mean curve: 0.0000
========================================
```

Testing the new function with TRUE == 1 AND PRED == 0/1s

```
true = pd.read_csv("~/AuprcRoc/data-fake/true_1.csv")
pred = pd.read_csv("~/AuprcRoc/data-fake/pred_2.csv")
fpr_dict, tpr_dict, macro_auc, macro_auc_interp, macro_auc_df = robust_macro_roc_auc(true, pred)
```

```
========================================
MACRO-AVERAGE ROC
========================================
Label1: True == 1 and PRED == 1 --> AUC = 1
```

```
Label2: True == 1 and PRED == 0 --> AUC = 0
Label3: True == 1 and PRED == 1 --> AUC = 1
Label4: True == 1 and PRED == 0 --> AUC = 0
Label5: True == 1 and PRED == 1 --> AUC = 1

----------------------------------------
Macro AUC mean of individual AUCs: 0.6000
Macro AUC interpolated mean curve: 0.3000
========================================
```

Testing the new function with TRUE == 1 AND PRED == 0/1s

```
true = pd.read_csv("~/AuprcRoc/data-fake/true_1.csv")
pred = pd.read_csv("~/AuprcRoc/data-fake/pred_3.csv")
fpr_dict, tpr_dict, macro_auc, macro_auc_interp, macro_auc_df =robust_macro_roc_auc(true, pred)
```

```
========================================
MACRO-AVERAGE ROC
========================================
Label1: True == 1 but PRED == 0/1 --> AUC = 1
Label2: True == 1 but PRED == 0/1 --> AUC = 1
Label3: True == 1 but PRED == 0/1 --> AUC = 1
Label4: True == 1 but PRED == 0/1 --> AUC = 1
Label5: True == 1 but PRED == 0/1 --> AUC = 1

----------------------------------------
Macro AUC mean of individual AUCs: 1.0000
Macro AUC interpolated mean curve: 0.5000
========================================
```

Testing the new function with TRUE == 1 AND PRED == 0/1

```
true = pd.read_csv("~/AuprcRoc/data-fake/true_1.csv")
pred = pd.read_csv("~/AuprcRoc/data-fake/pred_4.csv")
fpr_dict, tpr_dict, macro_auc, macro_auc_interp, macro_auc_df = robust_macro_roc_auc(true, pred)
```

```
========================================
MACRO-AVERAGE ROC
========================================
Label1: True == 1 but PRED == 0/1 --> AUC = 1
Label2: True == 1 but PRED == 0/1 --> AUC = 1
Label3: True == 1 but PRED == 0/1 --> AUC = 1
Label4: True == 1 but PRED == 0/1 --> AUC = 1
Label5: True == 1 but PRED == 0/1 --> AUC = 1
```

```
----------------------------------------
Macro AUC mean of individual AUCs: 1.0000
Macro AUC interpolated mean curve: 0.5000
========================================
```

Testing the new function with TRUE == 1 AND PRED == probabilities

```
true = pd.read_csv("~/AuprcRoc/data-fake/true_1.csv")
pred = pd.read_csv("~/AuprcRoc/data-fake/pred_5.csv")
fpr_dict, tpr_dict, macro_auc, macro_auc_interp, macro_auc_df = robust_macro_roc_auc(true, pred)
```

```
========================================
MACRO-AVERAGE ROC
========================================
Label1: True == 1 and PRED probabilistic --> AUC = 0.5
Label2: True == 1 and PRED probabilistic --> AUC = 0.5
Label3: True == 1 and PRED probabilistic --> AUC = 0.5
Label4: True == 1 and PRED probabilistic --> AUC = 0.5
Label5: True == 1 and PRED probabilistic --> AUC = 0.5

----------------------------------------
Macro AUC mean of individual AUCs: 0.5000
Macro AUC interpolated mean curve: 0.5000
========================================
```

Testing the new function with TRUE == 0/1 AND PRED == probabilities

```
true = pd.read_csv("~/AuprcRoc/data-fake/true_2.csv")
pred = pd.read_csv("~/AuprcRoc/data-fake/pred_5.csv")
fpr_dict, tpr_dict, macro_auc, macro_auc_interp, macro_auc_df =robust_macro_roc_auc(true, pred)
```

```
========================================
MACRO-AVERAGE ROC
========================================
Label1: True == 1 and PRED probabilistic --> AUC = 0.5
Label2: True == 0 and PRED probabilistic --> AUC = 0.5
Label3: True == 1 and PRED probabilistic --> AUC = 0.5
Label4: True == 0 and PRED probabilistic --> AUC = 0.5
Label5: True == 1 and PRED probabilistic --> AUC = 0.5

----------------------------------------
Macro AUC mean of individual AUCs: 0.5000
Macro AUC interpolated mean curve: 0.5000
========================================
```

Testing the new function with TRUE == 0/1 AND PRED == probabilities

```
true = pd.read_csv("~/AuprcRoc/data-fake/true_3.csv")
pred = pd.read_csv("~/AuprcRoc/data-fake/pred_5.csv")
fpr_dict, tpr_dict, macro_auc, macro_auc_interp, macro_auc_df = robust_macro_roc_auc(true, pred)
```

```
========================================
MACRO-AVERAGE ROC
========================================
Label1: Normal case (AUC = 0.7600)
Label2: Normal case (AUC = 0.6000)
Label3: Normal case (AUC = 0.6800)
Label4: Normal case (AUC = 0.7600)
Label5: Normal case (AUC = 0.4000)

----------------------------------------
Macro AUC mean of individual AUCs: 0.6400
Macro AUC interpolated mean curve: 0.7240
========================================
```

Testing the new function with TRUE == 0/1 AND PRED == probabilities

```
true = pd.read_csv("~/AuprcRoc/data-fake/true_3.csv")
pred = pd.read_csv("~/AuprcRoc/data-fake/pred_5.csv")
roc_auc_score(true, pred)
```

```
0.6399999999999999
```

Testing the new function with TRUE == 0/1 AND PRED == probabilities

```
true = pd.read_csv("~/AuprcRoc/data-fake/true_4.csv")
pred = pd.read_csv("~/AuprcRoc/data-fake/pred_5.csv")
fpr_dict, tpr_dict, macro_auc, macro_auc_interp, macro_auc_df = robust_macro_roc_auc(true, pred)
```

```
========================================
MACRO-AVERAGE ROC
========================================
Label1: Normal case (AUC = 0.7600)
Label2: Normal case (AUC = 0.4000)
Label3: Normal case (AUC = 0.6800)
Label4: Normal case (AUC = 0.2400)
Label5: Normal case (AUC = 0.4000)

----------------------------------------
```

```
Macro AUC mean of individual AUCs: 0.4960
Macro AUC interpolated mean curve: 0.5720
========================================
```

Computing with the original roc auc score from sckit learn:

```
true = pd.read_csv("~/AuprcRoc/data-fake/true_4.csv")
pred = pd.read_csv("~/AuprcRoc/data-fake/pred_5.csv")
roc_auc_score(true, pred)
```

```
0.4960000000000001
```

Testing with a real true and pred labels with special cases:

```
true = pd.read_csv("~/AuprcRoc/data-real/GnegativeGO-results-lcc/Split-3/y_true.csv")
true
```

	Label1	Label2	Label3	Label4	Label5	Label6	Label7	Label8
0	1	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0
2	1	0	0	0	0	0	0	0
3	1	0	1	0	0	0	0	0
4	1	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...
135	0	0	0	0	0	0	0	1
136	0	0	0	0	0	0	0	1
137	0	0	0	0	0	0	0	1
138	0	0	0	0	0	0	0	1
139	0	0	0	0	0	0	0	1

```
pred = pd.read_csv("~/AuprcRoc/data-real/GnegativeGO-results-lcc/Split-3/y_proba.csv")
pred
```

	Label1	Label2	Label3	Label4	Label5	Label6	Label7	Label8
0	1.000000	0.00	0.000	0.000	0.0	0.0	0.0	0.000
1	1.000000	0.00	0.000	0.000	0.0	0.0	0.0	0.000
2	1.000000	0.00	0.000	0.000	0.0	0.0	0.0	0.000
3	0.793333	0.01	0.500	0.000	0.0	0.0	0.0	0.005
4	0.960000	0.04	0.010	0.000	0.0	0.0	0.0	0.000
...	...	...	...	...	...	...	...	...
135	0.010000	0.00	0.000	0.015	0.0	0.0	0.0	0.935
136	0.000000	0.00	0.005	0.045	0.0	0.0	0.0	0.940
137	0.000000	0.00	0.000	0.000	0.0	0.0	0.0	1.000
138	0.000000	0.00	0.595	0.000	0.0	0.0	0.0	0.655

	Label1	Label2	Label3	Label4	Label5	Label6	Label7	Label8
139	0.000000	0.00	0.000	0.000	0.0	0.0	0.0	0.985

Original macro auc

```
roc_auc_score(true, pred)
```

```
/home/cissagatto/miniforge3/envs/ELCC/lib/python3.10/site-packages/sklearn/metrics/_ranking.py:424:
  warnings.warn(
```

```
nan
```

```
roc_auc_score(true, pred, average='macro')
```

```
/home/cissagatto/miniforge3/envs/ELCC/lib/python3.10/site-packages/sklearn/metrics/_ranking.py:424:
  warnings.warn(
```

```
nan
```

Robust macro auc

```
fpr_dict, tpr_dict, macro_auc, macro_auc_interp, macro_auc_df = robust_macro_roc_auc(true, pred)
```

```
========================================
MACRO-AVERAGE ROC
========================================
Label1: Normal case (AUC = 0.9991)
Label2: Normal case (AUC = 1.0000)
Label3: Normal case (AUC = 0.9978)
Label4: Normal case (AUC = 0.9982)
Label5: Normal case (AUC = 0.9963)
Label6: Normal case (AUC = 1.0000)
Label7: True == 0 and PRED probabilistic --> AUC = 0.5
Label8: Normal case (AUC = 0.9982)

----------------------------------------
Macro AUC mean of individual AUCs: 0.9362
Macro AUC interpolated mean curve: 0.9363
========================================
```

3.4. Final Roc Auc Micro

The Micro-average ROC-AUC combines all classes and samples into a single set of decisions, treating each (sample, class) pair as an independent observation. Unlike macro-average, which calculates an AUC per class and then averages them, micro-average calculates a single global ROC curve from all concatenated labels. Thus, each instance has the same weight in the metric, which makes micro-average more suitable for imbalanced datasets, where some classes have many more examples than others.

```
from sklearn import metrics
import numpy as np


def robust_micro_roc_auc(true, pred, verbose=True):
    """
    Compute a robust micro-average ROC-AUC score with special-case handling.

    ------------------------------------------------------------
    Description
    ------------------------------------------------------------
    This function calculates the micro-averaged ROC-AUC score for multilabel
    classification tasks, with robust handling of degenerate cases such as:
    - All labels being 0 or 1.
    - Model predicting a single constant value.
    - Lack of positive or negative samples.
    - Probabilistic predictions when the ground truth is constant.

    Unlike `sklearn.metrics.roc_auc_score`, this implementation avoids NaN
    or undefined results by explicitly assigning AUC values (0.0, 0.5, or 1.0)
    in these special scenarios.

    ------------------------------------------------------------
    Objective
    ------------------------------------------------------------
    Provide a consistent and interpretable AUC_micro value even when
    some edge cases occur - particularly useful in highly imbalanced
    multilabel problems or small datasets.

    ------------------------------------------------------------
    Parameters
    ------------------------------------------------------------
    true : pandas.DataFrame
        Ground-truth binary labels of shape (n_samples, n_classes).

    pred : pandas.DataFrame
        Predicted probabilities or binary predictions of the same shape.

    verbose : bool, optional (default=True)
        If True, prints diagnostic messages about which special case
        was applied and the computed AUC value.
```

```
    ------------------------------------------------------------
    Returns
    ------------------------------------------------------------
    fpr_micro : numpy.ndarray
        False Positive Rate values for the micro-averaged ROC curve.

    tpr_micro : numpy.ndarray
        True Positive Rate values for the micro-averaged ROC curve.

    auc_micro : float
        The computed micro-average AUC value. Guaranteed to be numeric
        and well-defined (never NaN).

    ------------------------------------------------------------
    Example
    ------------------------------------------------------------
    >>> import pandas as pd
    >>> true = pd.DataFrame({
    ...     'Label1': [0, 1, 1, 0],
    ...     'Label2': [1, 0, 1, 0]
    ... })
    >>> pred = pd.DataFrame({
    ...     'Label1': [0.1, 0.9, 0.8, 0.2],
    ...     'Label2': [0.7, 0.3, 0.9, 0.1]
    ... })
    >>> fpr_micro, tpr_micro, auc_micro = robust_micro_roc_auc(true, pred)
    >>> print(f"Micro-average AUC: {auc_micro:.4f}")
    ------------------------------------------------------------
    """

    # ------------------------------------------------------------------------
    # Flatten all labels into a single vector (treat every (sample, label) pair)
    # ------------------------------------------------------------------------
    y_true_all = true.values.ravel().astype(float)
    y_pred_all = pred.values.ravel().astype(float)

    if verbose:
        print("\n" + "=" * 40)
        print("MICRO-AVERAGE ROC")
        print("=" * 40)

    # --- CASE 1: y_true has only one class (no variation) ---
    if len(np.unique(y_true_all)) < 2:

        # (i) both empty (True=0, Pred=0)
        if np.all(y_true_all == 0) and np.all(y_pred_all == 0):
            auc_micro = 1.0
            fpr_micro = np.array([0.0, 1.0])
```

```
        tpr_micro = np.array([0.0, 1.0])
        msg = "True == 0 and PRED == 0 --> AUC = 1"

    # (ii) both full (True=1, Pred=1)
    elif np.all(y_true_all == 1) and np.all(y_pred_all == 1):
        auc_micro = 1.0
        fpr_micro = np.array([0.0, 1.0])
        tpr_micro = np.array([0.0, 1.0])
        msg = "True == 1 and PRED == 1 --> AUC = 1"

    # (iii) True empty, Pred full
    elif np.all(y_true_all == 0) and np.all(y_pred_all == 1):
        auc_micro = 0.0
        fpr_micro = np.array([0.0, 1.0])
        tpr_micro = np.array([1.0, 1.0])
        msg = "True == 0 and PRED == 1 --> AUC = 0"

    # (iv) True full, Pred empty
    elif np.all(y_true_all == 1) and np.all(y_pred_all == 0):
        auc_micro = 0.0
        fpr_micro = np.array([0.0, 1.0])
        tpr_micro = np.array([0.0, 0.0])
        msg = "True == 1 and PRED == 0 --> AUC = 0"

    # (v) True constant (0 or 1), but PRED has probabilities (not 0/1)
    elif len(np.unique(y_true_all)) == 1 and not np.all(np.isin(y_pred_all, [0, 1])):
        auc_micro = 0.5
        fpr_micro = np.array([0.0, 1.0])
        tpr_micro = np.array([0.0, 1.0])
        true_val = int(y_true_all[0])
        msg = f"True == {true_val} and PRED probabilistic --> AUC = 0.5"

    # (vi) True all 1, Pred mixed (0/1)
    elif np.all(y_true_all == 1) and np.any(y_pred_all == 0) and np.any(y_pred_all == 1):
        auc_micro = 1.0
        fpr_micro = np.array([0.0, 1.0])
        tpr_micro = np.array([0.0, 1.0])
        msg = "True == 1 but PRED == 0/1 --> AUC = 1"

    # (vii) True all 0, Pred mixed (0/1)
    elif np.all(y_true_all == 0) and np.any(y_pred_all == 0) and np.any(y_pred_all == 1):
        auc_micro = 0.0
        fpr_micro = np.array([0.0, 1.0])
        tpr_micro = np.array([0.0, 0.0])
        msg = "True == 0 but PRED == 0/1 --> AUC = 0"

    # fallback (shouldn't happen)
    else:
```

```
            auc_micro = 0.5
            fpr_micro = np.array([0.0, 1.0])
            tpr_micro = np.array([0.0, 1.0])
            true_val = int(y_true_all[0])
            msg = f"Other special case (True == {true_val}) --> AUC = 0.5"

        if verbose:
            print(msg)

    # --- CASE 2: normal ROC ---
    else:
        try:
            fpr_micro, tpr_micro, _ = metrics.roc_curve(y_true_all, y_pred_all)
            auc_micro = metrics.auc(fpr_micro, tpr_micro)
            if verbose:
                print(f"Normal case (AUC = {auc_micro:.4f})")
        except ValueError:
            fpr_micro = np.array([0.0, 1.0])
            tpr_micro = np.array([0.0, 0.0])
            auc_micro = 0.0
            if verbose:
                print("ROC computation failed --> AUC = 0")

    if verbose:
        print("\n" + "-" * 40)
        print(f"Micro AUC: {auc_micro:.4f}")
        print("=" * 40)

    return fpr_micro, tpr_micro, auc_micro
```

Testing with a real true and pred labels with special cases:

```
true = pd.read_csv("~/AuprcRoc/data-real/GnegativeGO-results-lcc/Split-3/y_true.csv")
true
```

	Label1	Label2	Label3	Label4	Label5	Label6	Label7	Label8
0	1	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0
2	1	0	0	0	0	0	0	0
3	1	0	1	0	0	0	0	0
4	1	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...
135	0	0	0	0	0	0	0	1
136	0	0	0	0	0	0	0	1
137	0	0	0	0	0	0	0	1
138	0	0	0	0	0	0	0	1
139	0	0	0	0	0	0	0	1

```
pred = pd.read_csv("~/AuprcRoc/data-real/GnegativeGO-results-lcc/Split-3/y_proba.csv")
pred
```

	Label1	Label2	Label3	Label4	Label5	Label6	Label7	Label8
0	1.000000	0.00	0.000	0.000	0.0	0.0	0.0	0.000
1	1.000000	0.00	0.000	0.000	0.0	0.0	0.0	0.000
2	1.000000	0.00	0.000	0.000	0.0	0.0	0.0	0.000
3	0.793333	0.01	0.500	0.000	0.0	0.0	0.0	0.005
4	0.960000	0.04	0.010	0.000	0.0	0.0	0.0	0.000
...	...	...	...	...	...	...	...	...
135	0.010000	0.00	0.000	0.015	0.0	0.0	0.0	0.935
136	0.000000	0.00	0.005	0.045	0.0	0.0	0.0	0.940
137	0.000000	0.00	0.000	0.000	0.0	0.0	0.0	1.000
138	0.000000	0.00	0.595	0.000	0.0	0.0	0.0	0.655
139	0.000000	0.00	0.000	0.000	0.0	0.0	0.0	0.985

Original micro auc

```
roc_auc_score(true, pred)
```

```
/home/cissagatto/miniforge3/envs/ELCC/lib/python3.10/site-packages/sklearn/metrics/_ranking.py:424:
  warnings.warn(
```

```
nan
```

```
roc_auc_score(true, pred, average='micro')
```

```
0.999158267020336
```

```
frp, tpr, auc_micro = robust_micro_roc_auc(true, pred)
```

```
========================================
MICRO-AVERAGE ROC
========================================
Normal case (AUC = 0.9992)

----------------------------------------
Micro AUC: 0.9992
========================================
```

Testing the new function with TRUE == 0 AND PRED == 0

```
true = pd.read_csv("~/AuprcRoc/data-fake/true_0.csv")
pred = pd.read_csv("~/AuprcRoc/data-fake/pred_0.csv")
fpr, tpr, micro_auc = robust_micro_roc_auc(true, pred)
```

```
========================================
MICRO-AVERAGE ROC
========================================
True == 0 and PRED == 0 --> AUC = 1

----------------------------------------
Micro AUC: 1.0000
========================================
```

```
fpr
```

```
array([0., 1.])
```

```
tpr
```

```
array([0., 1.])
```

```
micro_auc
```

```
1.0
```

Testing the new function with TRUE == 0 AND PRED == 1

```
true = pd.read_csv("~/AuprcRoc/data-fake/true_0.csv")
pred = pd.read_csv("~/AuprcRoc/data-fake/pred_1.csv")
fpr, tpr, micro_auc = robust_micro_roc_auc(true, pred)
```

```
========================================
MICRO-AVERAGE ROC
========================================
True == 0 and PRED == 1 --> AUC = 0

----------------------------------------
Micro AUC: 0.0000
========================================
```

Testing the new function with TRUE == 0 AND PRED == 1/0s

```
true = pd.read_csv("~/AuprcRoc/data-fake/true_0.csv")
pred = pd.read_csv("~/AuprcRoc/data-fake/pred_2.csv")
fpr, tpr, micro_auc = robust_micro_roc_auc(true, pred)
```

```
========================================
MICRO-AVERAGE ROC
========================================
True == 0 but PRED == 0/1 --> AUC = 0

----------------------------------------
Micro AUC: 0.0000
========================================
```

Testing the new function with TRUE == 0 AND PRED == 1/0s

```
true = pd.read_csv("~/AuprcRoc/data-fake/true_0.csv")
pred = pd.read_csv("~/AuprcRoc/data-fake/pred_3.csv")
fpr, tpr, micro_auc = robust_micro_roc_auc(true, pred)
```

```
========================================
MICRO-AVERAGE ROC
========================================
True == 0 but PRED == 0/1 --> AUC = 0

----------------------------------------
Micro AUC: 0.0000
========================================
```

Testing the new function with TRUE == 0 AND PRED == 1/0s

```
true = pd.read_csv("~/AuprcRoc/data-fake/true_0.csv")
pred = pd.read_csv("~/AuprcRoc/data-fake/pred_4.csv")
fpr, tpr, micro_auc = robust_micro_roc_auc(true, pred)
```

```
========================================
MICRO-AVERAGE ROC
========================================
True == 0 but PRED == 0/1 --> AUC = 0

----------------------------------------
Micro AUC: 0.0000
========================================
```

Testing the new function with TRUE == 0 AND PRED == probabilities

```
true = pd.read_csv("~/AuprcRoc/data-fake/true_0.csv")
pred = pd.read_csv("~/AuprcRoc/data-fake/pred_5.csv")
fpr, tpr, micro_auc = robust_micro_roc_auc(true, pred)
```

```
========================================
MICRO-AVERAGE ROC
========================================
True == 0 and PRED probabilistic --> AUC = 0.5

----------------------------------------
Micro AUC: 0.5000
========================================
```

Testing the new function with TRUE == 1 AND PRED == 1

```
true = pd.read_csv("~/AuprcRoc/data-fake/true_1.csv")
pred = pd.read_csv("~/AuprcRoc/data-fake/pred_1.csv")
fpr, tpr, micro_auc = robust_micro_roc_auc(true, pred)
```

```
========================================
MICRO-AVERAGE ROC
========================================
True == 1 and PRED == 1 --> AUC = 1

----------------------------------------
Micro AUC: 1.0000
========================================
```

Testing the new function with TRUE == 1 AND PRED == 0

```
true = pd.read_csv("~/AuprcRoc/data-fake/true_1.csv")
pred = pd.read_csv("~/AuprcRoc/data-fake/pred_0.csv")
fpr, tpr, micro_auc = robust_micro_roc_auc(true, pred)
```

```
========================================
MICRO-AVERAGE ROC
========================================
True == 1 and PRED == 0 --> AUC = 0

----------------------------------------
Micro AUC: 0.0000
========================================
```

Testing the new function with TRUE == 1 AND PRED == 0/1s

```
true = pd.read_csv("~/AuprcRoc/data-fake/true_1.csv")
pred = pd.read_csv("~/AuprcRoc/data-fake/pred_2.csv")
fpr, tpr, micro_auc = robust_micro_roc_auc(true, pred)
```

```
========================================
MICRO-AVERAGE ROC
========================================
True == 1 but PRED == 0/1 --> AUC = 1

----------------------------------------
Micro AUC: 1.0000
========================================
```

Testing the new function with TRUE == 1 AND PRED == 0/1s

```
true = pd.read_csv("~/AuprcRoc/data-fake/true_1.csv")
pred = pd.read_csv("~/AuprcRoc/data-fake/pred_3.csv")
fpr, tpr, micro_auc = robust_micro_roc_auc(true, pred)
```

```
========================================
MICRO-AVERAGE ROC
========================================
True == 1 but PRED == 0/1 --> AUC = 1

----------------------------------------
Micro AUC: 1.0000
========================================
```

Testing the new function with TRUE == 1 AND PRED == 0/1

```
true = pd.read_csv("~/AuprcRoc/data-fake/true_1.csv")
pred = pd.read_csv("~/AuprcRoc/data-fake/pred_4.csv")
fpr, tpr, micro_auc = robust_micro_roc_auc(true, pred)
```

```
========================================
MICRO-AVERAGE ROC
========================================
True == 1 but PRED == 0/1 --> AUC = 1

----------------------------------------
Micro AUC: 1.0000
========================================
```

Testing the new function with TRUE == 1 AND PRED == probabilities

```
true = pd.read_csv("~/AuprcRoc/data-fake/true_1.csv")
pred = pd.read_csv("~/AuprcRoc/data-fake/pred_5.csv")
fpr, tpr, micro_auc = robust_micro_roc_auc(true, pred)
```

```
========================================
MICRO-AVERAGE ROC
========================================
True == 1 and PRED probabilistic --> AUC = 0.5

----------------------------------------
Micro AUC: 0.5000
========================================
```

Testing the new function with TRUE == 0/1 AND PRED == probabilities

```
true = pd.read_csv("~/AuprcRoc/data-fake/true_2.csv")
pred = pd.read_csv("~/AuprcRoc/data-fake/pred_5.csv")
fpr, tpr, micro_auc = robust_micro_roc_auc(true, pred)
```

```
========================================
MICRO-AVERAGE ROC
========================================
Normal case (AUC = 0.5083)

----------------------------------------
Micro AUC: 0.5083
========================================
```

Testing the new function with TRUE == 0/1 AND PRED == probabilities

```
true = pd.read_csv("~/AuprcRoc/data-fake/true_3.csv")
pred = pd.read_csv("~/AuprcRoc/data-fake/pred_5.csv")
fpr, tpr, micro_auc = robust_micro_roc_auc(true, pred)
```

```
========================================
MICRO-AVERAGE ROC
========================================
Normal case (AUC = 0.6672)

----------------------------------------
Micro AUC: 0.6672
========================================
```

Testing the new function with TRUE == 0/1 AND PRED == probabilities

```
true = pd.read_csv("~/AuprcRoc/data-fake/true_4.csv")
pred = pd.read_csv("~/AuprcRoc/data-fake/pred_5.csv")
fpr, tpr, micro_auc = robust_micro_roc_auc(true, pred)
```

```
========================================
MICRO-AVERAGE ROC
========================================
Normal case (AUC = 0.4672)

----------------------------------------
Micro AUC: 0.4672
========================================
```

Testing the new function with TRUE == 0/1 AND PRED == probabilities

```
true = pd.read_csv("~/AuprcRoc/data-fake/true_4.csv")
pred = pd.read_csv("~/AuprcRoc/data-fake/pred_5.csv")
fpr, tpr, micro_auc = robust_micro_roc_auc(true, pred)
```

```
========================================
MICRO-AVERAGE ROC
========================================
Normal case (AUC = 0.4672)

----------------------------------------
Micro AUC: 0.4672
========================================
```

3.5. Final Roc Auc Samples

The samples-averaging method calculates the AUC by considering each sample (row) individually in a multilabel problem. Instead of combining all labels and instances as in micro-averaging, the calculation is done separately for each instance: each row is treated as a set of binary predictions, and the AUC is obtained by comparing the predictions and true values of that sample. Finally, the average of the AUCs of all samples is computed. This approach is useful when you want to measure the model's performance in a balanced way across instances, especially in unbalanced or multi-label datasets.

```
from sklearn import metrics
import numpy as np
import pandas as pd

def robust_sample_roc_auc(true, pred, verbose=True):
```

```
    """
    Compute a robust sample-average ROC-AUC with special case handling and FPR/TPR tracking.

    ----------------------------------------------------------------------------
    Description
    ----------------------------------------------------------------------------
    This function computes the **samples-average ROC-AUC**, which evaluates
    the ROC-AUC per sample (row) and then averages the results.
    It is particularly relevant for multi-label classification, where each
    sample can belong to multiple classes.

    Each sample's ROC curve (FPR, TPR) is computed individually.
    The function also handles degenerate cases robustly:
        - When all true labels or predictions for a sample are constant
        - When both are all zeros or all ones (perfect cases)
        - When true and predicted values are mismatched (inverted cases)

    ----------------------------------------------------------------------------
    Objective
    ----------------------------------------------------------------------------
    To compute a stable and interpretable per-sample ROC-AUC and summarize
    both the individual and average results, even when some samples have
    degenerate or constant values.

    ----------------------------------------------------------------------------
    Parameters
    ----------------------------------------------------------------------------
    true : pandas.DataFrame
        Binary ground-truth labels (0 or 1) for each class per sample.

    pred : pandas.DataFrame
        Predicted probabilities or binary predictions (0 or 1)
        with the same structure as `true`.

    verbose : bool, default=True
        If True, prints detailed information for each sample and overall stats.

    ----------------------------------------------------------------------------
    Returns
    ----------------------------------------------------------------------------
    sample_auc_df : pandas.DataFrame
        DataFrame containing per-sample ROC-AUC results:
            - "Sample": name of the sample (e.g., Sample1, Sample2, ...)
            - "AUC": computed ROC-AUC value
            - "FPR": False Positive Rate array
            - "TPR": True Positive Rate array

    samples_auc_mean : float
```

```
    Mean AUC across all samples.

--------------------------------------------------------------------------------
Example
--------------------------------------------------------------------------------
>>> import pandas as pd
>>> true = pd.DataFrame([[1, 0, 1], [0, 1, 0], [1, 1, 0]],
...                     columns=["Label1", "Label2", "Label3"])
>>> pred = pd.DataFrame([[0.9, 0.2, 0.8], [0.1, 0.7, 0.3], [0.8, 0.9, 0.1]],
...                     columns=["Label1", "Label2", "Label3"])
>>> sample_auc_df, samples_auc_mean = robust_sample_roc_auc(true, pred)
>>> print(sample_auc_df)
    Sample    AUC             FPR             TPR
0  Sample1  1.000  [0.0, 1.0]     [0.0, 1.0]
1  Sample2  1.000  [0.0, 1.0]     [0.0, 1.0]
2  Sample3  1.000  [0.0, 1.0]     [0.0, 1.0]
>>> print(f"Samples average AUC = {samples_auc_mean:.4f}")
"""

sample_results = []

if verbose:
    print("\n" + "="*40)
    print("SAMPLES-AVERAGE ROC")
    print("="*40)

for i in range(len(true)):
    y_true = true.iloc[i].values
    y_pred = pred.iloc[i].values

    # --- CASE 1: y_true has only one class (no variation) ---
    if len(np.unique(y_true)) < 2:

        # (i) both empty (True=0, Pred=0)
        if np.all(y_true == 0) and np.all(y_pred == 0):
            auc, fpr, tpr = 1.0, np.array([0.0, 1.0]), np.array([0.0, 1.0])
            msg = f"Sample {i+1}: True == 0 and Pred == 0 --> AUC = 1"

        # (ii) both full (True=1, Pred=1)
        elif np.all(y_true == 1) and np.all(y_pred == 1):
            auc, fpr, tpr = 1.0, np.array([0.0, 1.0]), np.array([0.0, 1.0])
            msg = f"Sample {i+1}: True == 1 and Pred == 1 --> AUC = 1"

        # (iii) True empty, Pred full
        elif np.all(y_true == 0) and np.all(y_pred == 1):
            auc, fpr, tpr = 0.0, np.array([0.0, 1.0]), np.array([1.0, 1.0])
            msg = f"Sample {i+1}: True == 0 and Pred == 1 --> AUC = 0"
```

```
            # (iv) True full, Pred empty
            elif np.all(y_true == 1) and np.all(y_pred == 0):
                auc, fpr, tpr = 0.0, np.array([0.0, 1.0]), np.array([0.0, 0.0])
                msg = f"Sample {i+1}: True == 1 and Pred == 0 --> AUC = 0"

            # (v) True constant (0 or 1), but PRED has probabilities (not 0/1)
            else:
                auc, fpr, tpr = 0.5, np.array([0.0, 1.0]), np.array([0.0, 1.0])
                true_val = int(y_true[0])  # vai ser 0 ou 1
                msg = f"Sample {i+1}: True == {true_val} and PRED probabilistic --> AUC = 0.5"

            if verbose:
                print(msg)
        else:
            try:
                fpr, tpr, _ = metrics.roc_curve(y_true.astype(float), y_pred.astype(float))
                auc = metrics.auc(fpr, tpr)
                if verbose:
                    print(f"Sample {i+1}: Normal case (AUC = {auc:.4f})")
            except ValueError:
                auc, fpr, tpr = 0.0, np.array([0.0, 1.0]), np.array([0.0, 0.0])
                if verbose:
                    print(f"Sample {i+1}: ROC computation failed → AUC = 0")

        sample_results.append((i, auc, fpr, tpr))

    # Build final DataFrame
    sample_auc_df = pd.DataFrame(sample_results, columns=["Index", "AUC", "FPR", "TPR"])
    sample_auc_df["Sample"] = ["Sample" + str(i + 1) for i in sample_auc_df["Index"]]
    sample_auc_df = sample_auc_df[["Sample", "AUC", "FPR", "TPR"]]

    samples_auc_mean = np.mean(sample_auc_df["AUC"])

    if verbose:
        print("\n" + "-"*40)
        print(f"Samples mean AUC: {samples_auc_mean:.4f}")
        print("="*40)

    return sample_auc_df, samples_auc_mean
```

Testing with a real true and pred labels with special cases:

```
true = pd.read_csv("~/AuprcRoc/data-real/GnegativeGO-results-lcc/Split-3/y_true.csv")
true
```

	Label1	Label2	Label3	Label4	Label5	Label6	Label7	Label8
0	1	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0
2	1	0	0	0	0	0	0	0
3	1	0	1	0	0	0	0	0
4	1	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...
135	0	0	0	0	0	0	0	1
136	0	0	0	0	0	0	0	1
137	0	0	0	0	0	0	0	1
138	0	0	0	0	0	0	0	1
139	0	0	0	0	0	0	0	1

```
pred = pd.read_csv("~/AuprcRoc/data-real/GnegativeGO-results-lcc/Split-3/y_proba.csv")
pred
```

	Label1	Label2	Label3	Label4	Label5	Label6	Label7	Label8
0	1.000000	0.00	0.000	0.000	0.0	0.0	0.0	0.000
1	1.000000	0.00	0.000	0.000	0.0	0.0	0.0	0.000
2	1.000000	0.00	0.000	0.000	0.0	0.0	0.0	0.000
3	0.793333	0.01	0.500	0.000	0.0	0.0	0.0	0.005
4	0.960000	0.04	0.010	0.000	0.0	0.0	0.0	0.000
...	...	...	...	...	...	...	...	...
135	0.010000	0.00	0.000	0.015	0.0	0.0	0.0	0.935
136	0.000000	0.00	0.005	0.045	0.0	0.0	0.0	0.940
137	0.000000	0.00	0.000	0.000	0.0	0.0	0.0	1.000
138	0.000000	0.00	0.595	0.000	0.0	0.0	0.0	0.655
139	0.000000	0.00	0.000	0.000	0.0	0.0	0.0	0.985

```
roc_auc_score(true, pred)
```

```
/home/cissagatto/miniforge3/envs/ELCC/lib/python3.10/site-packages/sklearn/metrics/_ranking.py:424:
  warnings.warn(
```

```
nan
```

```
roc_auc_score(true, pred, average='samples')
```

```
0.9969387755102042
```

```
sample_auc_df, samples_auc_mean = robust_sample_roc_auc(true, pred)
```

```
======================================
SAMPLES-AVERAGE ROC
======================================
Sample 1: Normal case (AUC = 1.0000)
Sample 2: Normal case (AUC = 1.0000)
Sample 3: Normal case (AUC = 1.0000)
Sample 4: Normal case (AUC = 1.0000)
Sample 5: Normal case (AUC = 1.0000)
Sample 6: Normal case (AUC = 1.0000)
Sample 7: Normal case (AUC = 1.0000)
Sample 8: Normal case (AUC = 1.0000)
Sample 9: Normal case (AUC = 1.0000)
Sample 10: Normal case (AUC = 1.0000)
Sample 11: Normal case (AUC = 1.0000)
Sample 12: Normal case (AUC = 1.0000)
Sample 13: Normal case (AUC = 1.0000)
Sample 14: Normal case (AUC = 1.0000)
Sample 15: Normal case (AUC = 1.0000)
Sample 16: Normal case (AUC = 1.0000)
Sample 17: Normal case (AUC = 1.0000)
Sample 18: Normal case (AUC = 1.0000)
Sample 19: Normal case (AUC = 1.0000)
Sample 20: Normal case (AUC = 1.0000)
Sample 21: Normal case (AUC = 1.0000)
Sample 22: Normal case (AUC = 1.0000)
Sample 23: Normal case (AUC = 1.0000)
Sample 24: Normal case (AUC = 1.0000)
Sample 25: Normal case (AUC = 1.0000)
Sample 26: Normal case (AUC = 1.0000)
Sample 27: Normal case (AUC = 1.0000)
Sample 28: Normal case (AUC = 1.0000)
Sample 29: Normal case (AUC = 1.0000)
Sample 30: Normal case (AUC = 1.0000)
Sample 31: Normal case (AUC = 1.0000)
Sample 32: Normal case (AUC = 1.0000)
Sample 33: Normal case (AUC = 1.0000)
Sample 34: Normal case (AUC = 1.0000)
Sample 35: Normal case (AUC = 1.0000)
Sample 36: Normal case (AUC = 1.0000)
Sample 37: Normal case (AUC = 1.0000)
Sample 38: Normal case (AUC = 1.0000)
Sample 39: Normal case (AUC = 1.0000)
Sample 40: Normal case (AUC = 1.0000)
Sample 41: Normal case (AUC = 1.0000)
Sample 42: Normal case (AUC = 1.0000)
Sample 43: Normal case (AUC = 1.0000)
Sample 44: Normal case (AUC = 1.0000)
Sample 45: Normal case (AUC = 1.0000)
```

```
Sample 46: Normal case (AUC = 1.0000)
Sample 47: Normal case (AUC = 1.0000)
Sample 48: Normal case (AUC = 1.0000)
Sample 49: Normal case (AUC = 1.0000)
Sample 50: Normal case (AUC = 1.0000)
Sample 51: Normal case (AUC = 1.0000)
Sample 52: Normal case (AUC = 1.0000)
Sample 53: Normal case (AUC = 1.0000)
Sample 54: Normal case (AUC = 1.0000)
Sample 55: Normal case (AUC = 1.0000)
Sample 56: Normal case (AUC = 1.0000)
Sample 57: Normal case (AUC = 1.0000)
Sample 58: Normal case (AUC = 1.0000)
Sample 59: Normal case (AUC = 1.0000)
Sample 60: Normal case (AUC = 1.0000)
Sample 61: Normal case (AUC = 1.0000)
Sample 62: Normal case (AUC = 1.0000)
Sample 63: Normal case (AUC = 1.0000)
Sample 64: Normal case (AUC = 1.0000)
Sample 65: Normal case (AUC = 1.0000)
Sample 66: Normal case (AUC = 1.0000)
Sample 67: Normal case (AUC = 1.0000)
Sample 68: Normal case (AUC = 1.0000)
Sample 69: Normal case (AUC = 1.0000)
Sample 70: Normal case (AUC = 1.0000)
Sample 71: Normal case (AUC = 1.0000)
Sample 72: Normal case (AUC = 1.0000)
Sample 73: Normal case (AUC = 0.8571)
Sample 74: Normal case (AUC = 1.0000)
Sample 75: Normal case (AUC = 1.0000)
Sample 76: Normal case (AUC = 1.0000)
Sample 77: Normal case (AUC = 1.0000)
Sample 78: Normal case (AUC = 1.0000)
Sample 79: Normal case (AUC = 1.0000)
Sample 80: Normal case (AUC = 1.0000)
Sample 81: Normal case (AUC = 1.0000)
Sample 82: Normal case (AUC = 1.0000)
Sample 83: Normal case (AUC = 1.0000)
Sample 84: Normal case (AUC = 1.0000)
Sample 85: Normal case (AUC = 1.0000)
Sample 86: Normal case (AUC = 1.0000)
Sample 87: Normal case (AUC = 1.0000)
Sample 88: Normal case (AUC = 1.0000)
Sample 89: Normal case (AUC = 1.0000)
Sample 90: Normal case (AUC = 1.0000)
Sample 91: Normal case (AUC = 1.0000)
Sample 92: Normal case (AUC = 1.0000)
Sample 93: Normal case (AUC = 0.8571)
Sample 94: Normal case (AUC = 1.0000)
```

```
Sample 95: Normal case (AUC = 1.0000)
Sample 96: Normal case (AUC = 1.0000)
Sample 97: Normal case (AUC = 1.0000)
Sample 98: Normal case (AUC = 1.0000)
Sample 99: Normal case (AUC = 1.0000)
Sample 100: Normal case (AUC = 1.0000)
Sample 101: Normal case (AUC = 1.0000)
Sample 102: Normal case (AUC = 1.0000)
Sample 103: Normal case (AUC = 1.0000)
Sample 104: Normal case (AUC = 1.0000)
Sample 105: Normal case (AUC = 1.0000)
Sample 106: Normal case (AUC = 1.0000)
Sample 107: Normal case (AUC = 1.0000)
Sample 108: Normal case (AUC = 1.0000)
Sample 109: Normal case (AUC = 1.0000)
Sample 110: Normal case (AUC = 1.0000)
Sample 111: Normal case (AUC = 1.0000)
Sample 112: Normal case (AUC = 1.0000)
Sample 113: Normal case (AUC = 1.0000)
Sample 114: Normal case (AUC = 1.0000)
Sample 115: Normal case (AUC = 1.0000)
Sample 116: Normal case (AUC = 1.0000)
Sample 117: Normal case (AUC = 1.0000)
Sample 118: Normal case (AUC = 1.0000)
Sample 119: Normal case (AUC = 1.0000)
Sample 120: Normal case (AUC = 1.0000)
Sample 121: Normal case (AUC = 1.0000)
Sample 122: Normal case (AUC = 1.0000)
Sample 123: Normal case (AUC = 1.0000)
Sample 124: Normal case (AUC = 1.0000)
Sample 125: Normal case (AUC = 1.0000)
Sample 126: Normal case (AUC = 1.0000)
Sample 127: Normal case (AUC = 0.8571)
Sample 128: Normal case (AUC = 1.0000)
Sample 129: Normal case (AUC = 1.0000)
Sample 130: Normal case (AUC = 1.0000)
Sample 131: Normal case (AUC = 1.0000)
Sample 132: Normal case (AUC = 1.0000)
Sample 133: Normal case (AUC = 1.0000)
Sample 134: Normal case (AUC = 1.0000)
Sample 135: Normal case (AUC = 1.0000)
Sample 136: Normal case (AUC = 1.0000)
Sample 137: Normal case (AUC = 1.0000)
Sample 138: Normal case (AUC = 1.0000)
Sample 139: Normal case (AUC = 1.0000)
Sample 140: Normal case (AUC = 1.0000)

----------------------------------------
Samples mean AUC: 0.9969
```

```
========================================
```

```
samples_auc_mean
```

```
np.float64(0.9969387755102042)
```

```
sample_auc_df
```

	Sample	AUC	FPR	TPR
0	Sample1	1.0	[0.0, 0.0, 1.0]	[0.0, 1.0, 1.0]
1	Sample2	1.0	[0.0, 0.0, 1.0]	[0.0, 1.0, 1.0]
2	Sample3	1.0	[0.0, 0.0, 1.0]	[0.0, 1.0, 1.0]
3	Sample4	1.0	[0.0, 0.0, 0.0, 0.3333333333333333, 1.0]	[0.0, 0.5, 1.0, 1.0, 1.0]
4	Sample5	1.0	[0.0, 0.0, 0.2857142857142857, 1.0]	[0.0, 1.0, 1.0, 1.0]
...	...	...	...	...
135	Sample136	1.0	[0.0, 0.0, 0.2857142857142857, 1.0]	[0.0, 1.0, 1.0, 1.0]
136	Sample137	1.0	[0.0, 0.0, 0.2857142857142857, 1.0]	[0.0, 1.0, 1.0, 1.0]
137	Sample138	1.0	[0.0, 0.0, 1.0]	[0.0, 1.0, 1.0]
138	Sample139	1.0	[0.0, 0.0, 0.14285714285714285, 1.0]	[0.0, 1.0, 1.0, 1.0]
139	Sample140	1.0	[0.0, 0.0, 1.0]	[0.0, 1.0, 1.0]

Testing the new function with TRUE == 0 AND PRED == 0

```
true = pd.read_csv("~/AuprcRoc/data-fake/true_0.csv")
pred = pd.read_csv("~/AuprcRoc/data-fake/pred_0.csv")
sample_auc_df, samples_auc_mean = robust_sample_roc_auc(true, pred)
```

```
========================================
SAMPLES-AVERAGE ROC
========================================
Sample 1: True == 0 and Pred == 0 --> AUC = 1
Sample 2: True == 0 and Pred == 0 --> AUC = 1
Sample 3: True == 0 and Pred == 0 --> AUC = 1
Sample 4: True == 0 and Pred == 0 --> AUC = 1
Sample 5: True == 0 and Pred == 0 --> AUC = 1
Sample 6: True == 0 and Pred == 0 --> AUC = 1
Sample 7: True == 0 and Pred == 0 --> AUC = 1
Sample 8: True == 0 and Pred == 0 --> AUC = 1
Sample 9: True == 0 and Pred == 0 --> AUC = 1
Sample 10: True == 0 and Pred == 0 --> AUC = 1

----------------------------------------
Samples mean AUC: 1.0000
========================================
```

Testing the new function with TRUE == 0 AND PRED == 1

```
true = pd.read_csv("~/AuprcRoc/data-fake/true_0.csv")
pred = pd.read_csv("~/AuprcRoc/data-fake/pred_1.csv")
sample_auc_df, samples_auc_mean = robust_sample_roc_auc(true, pred)
```

```
========================================
SAMPLES-AVERAGE ROC
========================================
Sample 1: True == 0 and Pred == 1 --> AUC = 0
Sample 2: True == 0 and Pred == 1 --> AUC = 0
Sample 3: True == 0 and Pred == 1 --> AUC = 0
Sample 4: True == 0 and Pred == 1 --> AUC = 0
Sample 5: True == 0 and Pred == 1 --> AUC = 0
Sample 6: True == 0 and Pred == 1 --> AUC = 0
Sample 7: True == 0 and Pred == 1 --> AUC = 0
Sample 8: True == 0 and Pred == 1 --> AUC = 0
Sample 9: True == 0 and Pred == 1 --> AUC = 0
Sample 10: True == 0 and Pred == 1 --> AUC = 0

----------------------------------------
Samples mean AUC: 0.0000
========================================
```

Testing the new function with TRUE == 0 AND PRED == 1/0s

```
true = pd.read_csv("~/AuprcRoc/data-fake/true_0.csv")
pred = pd.read_csv("~/AuprcRoc/data-fake/pred_2.csv")
sample_auc_df, samples_auc_mean = robust_sample_roc_auc(true, pred)
```

```
========================================
SAMPLES-AVERAGE ROC
========================================
Sample 1: True == 0 and PRED probabilistic --> AUC = 0.5
Sample 2: True == 0 and PRED probabilistic --> AUC = 0.5
Sample 3: True == 0 and PRED probabilistic --> AUC = 0.5
Sample 4: True == 0 and PRED probabilistic --> AUC = 0.5
Sample 5: True == 0 and PRED probabilistic --> AUC = 0.5
Sample 6: True == 0 and PRED probabilistic --> AUC = 0.5
Sample 7: True == 0 and PRED probabilistic --> AUC = 0.5
Sample 8: True == 0 and PRED probabilistic --> AUC = 0.5
Sample 9: True == 0 and PRED probabilistic --> AUC = 0.5
Sample 10: True == 0 and PRED probabilistic --> AUC = 0.5

----------------------------------------
```

```
Samples mean AUC: 0.5000
========================================
```

Testing the new function with TRUE == 0 AND PRED == 1/0s

```
true = pd.read_csv("~/AuprcRoc/data-fake/true_0.csv")
pred = pd.read_csv("~/AuprcRoc/data-fake/pred_3.csv")
sample_auc_df, samples_auc_mean = robust_sample_roc_auc(true, pred)
```

```
========================================
SAMPLES-AVERAGE ROC
========================================
Sample 1: True == 0 and PRED probabilistic --> AUC = 0.5
Sample 2: True == 0 and PRED probabilistic --> AUC = 0.5
Sample 3: True == 0 and PRED probabilistic --> AUC = 0.5
Sample 4: True == 0 and PRED probabilistic --> AUC = 0.5
Sample 5: True == 0 and PRED probabilistic --> AUC = 0.5
Sample 6: True == 0 and PRED probabilistic --> AUC = 0.5
Sample 7: True == 0 and PRED probabilistic --> AUC = 0.5
Sample 8: True == 0 and PRED probabilistic --> AUC = 0.5
Sample 9: True == 0 and PRED probabilistic --> AUC = 0.5
Sample 10: True == 0 and PRED probabilistic --> AUC = 0.5

----------------------------------------
Samples mean AUC: 0.5000
========================================
```

Testing the new function with TRUE == 0 AND PRED == 1/0s

```
true = pd.read_csv("~/AuprcRoc/data-fake/true_0.csv")
pred = pd.read_csv("~/AuprcRoc/data-fake/pred_4.csv")
sample_auc_df, samples_auc_mean = robust_sample_roc_auc(true, pred)
```

```
========================================
SAMPLES-AVERAGE ROC
========================================
Sample 1: True == 0 and Pred == 1 --> AUC = 0
Sample 2: True == 0 and Pred == 0 --> AUC = 1
Sample 3: True == 0 and Pred == 1 --> AUC = 0
Sample 4: True == 0 and Pred == 0 --> AUC = 1
Sample 5: True == 0 and Pred == 1 --> AUC = 0
Sample 6: True == 0 and Pred == 0 --> AUC = 1
Sample 7: True == 0 and Pred == 1 --> AUC = 0
Sample 8: True == 0 and Pred == 0 --> AUC = 1
Sample 9: True == 0 and Pred == 1 --> AUC = 0
```

```
Sample 10: True == 0 and Pred == 0 --> AUC = 1

----------------------------------------
Samples mean AUC: 0.5000
========================================
```

Testing the new function with TRUE == 0 AND PRED == probabilities

```
true = pd.read_csv("~/AuprcRoc/data-fake/true_0.csv")
pred = pd.read_csv("~/AuprcRoc/data-fake/pred_5.csv")
sample_auc_df, samples_auc_mean = robust_sample_roc_auc(true, pred)
```

```
========================================
SAMPLES-AVERAGE ROC
========================================
Sample 1: True == 0 and PRED probabilistic --> AUC = 0.5
Sample 2: True == 0 and PRED probabilistic --> AUC = 0.5
Sample 3: True == 0 and PRED probabilistic --> AUC = 0.5
Sample 4: True == 0 and PRED probabilistic --> AUC = 0.5
Sample 5: True == 0 and PRED probabilistic --> AUC = 0.5
Sample 6: True == 0 and PRED probabilistic --> AUC = 0.5
Sample 7: True == 0 and PRED probabilistic --> AUC = 0.5
Sample 8: True == 0 and PRED probabilistic --> AUC = 0.5
Sample 9: True == 0 and PRED probabilistic --> AUC = 0.5
Sample 10: True == 0 and PRED probabilistic --> AUC = 0.5

----------------------------------------
Samples mean AUC: 0.5000
========================================
```

Testing the new function with TRUE == 1 AND PRED == 1

```
true = pd.read_csv("~/AuprcRoc/data-fake/true_1.csv")
pred = pd.read_csv("~/AuprcRoc/data-fake/pred_1.csv")
sample_auc_df, samples_auc_mean = robust_sample_roc_auc(true, pred)
```

```
========================================
SAMPLES-AVERAGE ROC
========================================
Sample 1: True == 1 and Pred == 1 --> AUC = 1
Sample 2: True == 1 and Pred == 1 --> AUC = 1
Sample 3: True == 1 and Pred == 1 --> AUC = 1
Sample 4: True == 1 and Pred == 1 --> AUC = 1
Sample 5: True == 1 and Pred == 1 --> AUC = 1
Sample 6: True == 1 and Pred == 1 --> AUC = 1
```

```
Sample 7: True == 1 and Pred == 1 --> AUC = 1
Sample 8: True == 1 and Pred == 1 --> AUC = 1
Sample 9: True == 1 and Pred == 1 --> AUC = 1
Sample 10: True == 1 and Pred == 1 --> AUC = 1

----------------------------------------
Samples mean AUC: 1.0000
========================================
```

Testing the new function with TRUE == 1 AND PRED == 0

```
true = pd.read_csv("~/AuprcRoc/data-fake/true_1.csv")
pred = pd.read_csv("~/AuprcRoc/data-fake/pred_0.csv")
sample_auc_df, samples_auc_mean = robust_sample_roc_auc(true, pred)
```

```
========================================
SAMPLES-AVERAGE ROC
========================================
Sample 1: True == 1 and Pred == 0 --> AUC = 0
Sample 2: True == 1 and Pred == 0 --> AUC = 0
Sample 3: True == 1 and Pred == 0 --> AUC = 0
Sample 4: True == 1 and Pred == 0 --> AUC = 0
Sample 5: True == 1 and Pred == 0 --> AUC = 0
Sample 6: True == 1 and Pred == 0 --> AUC = 0
Sample 7: True == 1 and Pred == 0 --> AUC = 0
Sample 8: True == 1 and Pred == 0 --> AUC = 0
Sample 9: True == 1 and Pred == 0 --> AUC = 0
Sample 10: True == 1 and Pred == 0 --> AUC = 0

----------------------------------------
Samples mean AUC: 0.0000
========================================
```

Testing the new function with TRUE == 1 AND PRED == 0/1s

```
true = pd.read_csv("~/AuprcRoc/data-fake/true_1.csv")
pred = pd.read_csv("~/AuprcRoc/data-fake/pred_2.csv")
sample_auc_df, samples_auc_mean = robust_sample_roc_auc(true, pred)
```

```
========================================
SAMPLES-AVERAGE ROC
========================================
Sample 1: True == 1 and PRED probabilistic --> AUC = 0.5
Sample 2: True == 1 and PRED probabilistic --> AUC = 0.5
Sample 3: True == 1 and PRED probabilistic --> AUC = 0.5
```

```
Sample 4: True == 1 and PRED probabilistic --> AUC = 0.5
Sample 5: True == 1 and PRED probabilistic --> AUC = 0.5
Sample 6: True == 1 and PRED probabilistic --> AUC = 0.5
Sample 7: True == 1 and PRED probabilistic --> AUC = 0.5
Sample 8: True == 1 and PRED probabilistic --> AUC = 0.5
Sample 9: True == 1 and PRED probabilistic --> AUC = 0.5
Sample 10: True == 1 and PRED probabilistic --> AUC = 0.5

---------------------------------------
Samples mean AUC: 0.5000
=======================================
```

Testing the new function with TRUE == 1 AND PRED == 0/1s

```
true = pd.read_csv("~/AuprcRoc/data-fake/true_1.csv")
pred = pd.read_csv("~/AuprcRoc/data-fake/pred_3.csv")
sample_auc_df, samples_auc_mean = robust_sample_roc_auc(true, pred)
```

```
=======================================
SAMPLES-AVERAGE ROC
=======================================
Sample 1: True == 1 and PRED probabilistic --> AUC = 0.5
Sample 2: True == 1 and PRED probabilistic --> AUC = 0.5
Sample 3: True == 1 and PRED probabilistic --> AUC = 0.5
Sample 4: True == 1 and PRED probabilistic --> AUC = 0.5
Sample 5: True == 1 and PRED probabilistic --> AUC = 0.5
Sample 6: True == 1 and PRED probabilistic --> AUC = 0.5
Sample 7: True == 1 and PRED probabilistic --> AUC = 0.5
Sample 8: True == 1 and PRED probabilistic --> AUC = 0.5
Sample 9: True == 1 and PRED probabilistic --> AUC = 0.5
Sample 10: True == 1 and PRED probabilistic --> AUC = 0.5

---------------------------------------
Samples mean AUC: 0.5000
=======================================
```

Testing the new function with TRUE == 1 AND PRED == 0/1

```
true = pd.read_csv("~/AuprcRoc/data-fake/true_1.csv")
pred = pd.read_csv("~/AuprcRoc/data-fake/pred_4.csv")
sample_auc_df, samples_auc_mean = robust_sample_roc_auc(true, pred)
```

```
=======================================
SAMPLES-AVERAGE ROC
=======================================
```

```
Sample 1: True == 1 and Pred == 1 --> AUC = 1
Sample 2: True == 1 and Pred == 0 --> AUC = 0
Sample 3: True == 1 and Pred == 1 --> AUC = 1
Sample 4: True == 1 and Pred == 0 --> AUC = 0
Sample 5: True == 1 and Pred == 1 --> AUC = 1
Sample 6: True == 1 and Pred == 0 --> AUC = 0
Sample 7: True == 1 and Pred == 1 --> AUC = 1
Sample 8: True == 1 and Pred == 0 --> AUC = 0
Sample 9: True == 1 and Pred == 1 --> AUC = 1
Sample 10: True == 1 and Pred == 0 --> AUC = 0

--------------------------------------
Samples mean AUC: 0.5000
======================================
```

Testing the new function with TRUE == 1 AND PRED == probabilities

```
true = pd.read_csv("~/AuprcRoc/data-fake/true_1.csv")
pred = pd.read_csv("~/AuprcRoc/data-fake/pred_5.csv")
sample_auc_df, samples_auc_mean = robust_sample_roc_auc(true, pred)
```

```
======================================
SAMPLES-AVERAGE ROC
======================================
Sample 1: True == 1 and PRED probabilistic --> AUC = 0.5
Sample 2: True == 1 and PRED probabilistic --> AUC = 0.5
Sample 3: True == 1 and PRED probabilistic --> AUC = 0.5
Sample 4: True == 1 and PRED probabilistic --> AUC = 0.5
Sample 5: True == 1 and PRED probabilistic --> AUC = 0.5
Sample 6: True == 1 and PRED probabilistic --> AUC = 0.5
Sample 7: True == 1 and PRED probabilistic --> AUC = 0.5
Sample 8: True == 1 and PRED probabilistic --> AUC = 0.5
Sample 9: True == 1 and PRED probabilistic --> AUC = 0.5
Sample 10: True == 1 and PRED probabilistic --> AUC = 0.5

--------------------------------------
Samples mean AUC: 0.5000
======================================
```

Testing the new function with TRUE == 0/1 AND PRED == probabilities

```
true = pd.read_csv("~/AuprcRoc/data-fake/true_2.csv")
pred = pd.read_csv("~/AuprcRoc/data-fake/pred_5.csv")
sample_auc_df, samples_auc_mean = robust_sample_roc_auc(true, pred)
```

```
========================================
SAMPLES-AVERAGE ROC
========================================
Sample 1: Normal case (AUC = 0.5000)
Sample 2: Normal case (AUC = 0.1667)
Sample 3: Normal case (AUC = 0.6667)
Sample 4: Normal case (AUC = 0.0000)
Sample 5: Normal case (AUC = 0.8333)
Sample 6: Normal case (AUC = 0.1667)
Sample 7: Normal case (AUC = 0.5000)
Sample 8: Normal case (AUC = 0.6667)
Sample 9: Normal case (AUC = 0.6667)
Sample 10: Normal case (AUC = 0.3333)

----------------------------------------
Samples mean AUC: 0.4500
========================================
```

```
true = pd.read_csv("~/AuprcRoc/data-fake/true_2.csv")
pred = pd.read_csv("~/AuprcRoc/data-fake/pred_5.csv")
roc_auc_score(true, pred, average='samples')
```

```
0.45
```

Testing the new function with TRUE == 0/1 AND PRED == probabilities

```
true = pd.read_csv("~/AuprcRoc/data-fake/true_3.csv")
pred = pd.read_csv("~/AuprcRoc/data-fake/pred_5.csv")
sample_auc_df, samples_auc_mean = robust_sample_roc_auc(true, pred)
```

```
========================================
SAMPLES-AVERAGE ROC
========================================
Sample 1: Normal case (AUC = 0.5000)
Sample 2: Normal case (AUC = 0.8333)
Sample 3: Normal case (AUC = 0.6667)
Sample 4: Normal case (AUC = 1.0000)
Sample 5: Normal case (AUC = 0.8333)
Sample 6: Normal case (AUC = 0.8333)
Sample 7: Normal case (AUC = 0.5000)
Sample 8: Normal case (AUC = 0.3333)
Sample 9: Normal case (AUC = 0.6667)
Sample 10: Normal case (AUC = 0.6667)

----------------------------------------
Samples mean AUC: 0.6833
========================================
```

```
true = pd.read_csv("~/AuprcRoc/data-fake/true_3.csv")
pred = pd.read_csv("~/AuprcRoc/data-fake/pred_5.csv")
roc_auc_score(true, pred, average='samples')
```

```
0.6833333333333333
```

Testing the new function with TRUE == 0/1 AND PRED == probabilities

```
true = pd.read_csv("~/AuprcRoc/data-fake/true_4.csv")
pred = pd.read_csv("~/AuprcRoc/data-fake/pred_5.csv")
sample_auc_df, samples_auc_mean = robust_sample_roc_auc(true, pred)
```

```
========================================
SAMPLES-AVERAGE ROC
========================================
Sample 1: True == 1 and PRED probabilistic --> AUC = 0.5
Sample 2: True == 0 and PRED probabilistic --> AUC = 0.5
Sample 3: True == 1 and PRED probabilistic --> AUC = 0.5
Sample 4: True == 0 and PRED probabilistic --> AUC = 0.5
Sample 5: True == 1 and PRED probabilistic --> AUC = 0.5
Sample 6: True == 0 and PRED probabilistic --> AUC = 0.5
Sample 7: True == 1 and PRED probabilistic --> AUC = 0.5
Sample 8: True == 0 and PRED probabilistic --> AUC = 0.5
Sample 9: True == 1 and PRED probabilistic --> AUC = 0.5
Sample 10: True == 0 and PRED probabilistic --> AUC = 0.5

----------------------------------------
Samples mean AUC: 0.5000
========================================
```

Testing the new function with TRUE == 0/1 AND PRED == probabilities

```
true = pd.read_csv("~/AuprcRoc/data-fake/true_4.csv")
pred = pd.read_csv("~/AuprcRoc/data-fake/pred_5.csv")
sample_auc_df, samples_auc_mean = robust_sample_roc_auc(true, pred)
```

```
========================================
SAMPLES-AVERAGE ROC
========================================
Sample 1: True == 1 and PRED probabilistic --> AUC = 0.5
Sample 2: True == 0 and PRED probabilistic --> AUC = 0.5
Sample 3: True == 1 and PRED probabilistic --> AUC = 0.5
Sample 4: True == 0 and PRED probabilistic --> AUC = 0.5
Sample 5: True == 1 and PRED probabilistic --> AUC = 0.5
```

```
Sample 6: True == 0 and PRED probabilistic --> AUC = 0.5
Sample 7: True == 1 and PRED probabilistic --> AUC = 0.5
Sample 8: True == 0 and PRED probabilistic --> AUC = 0.5
Sample 9: True == 1 and PRED probabilistic --> AUC = 0.5
Sample 10: True == 0 and PRED probabilistic --> AUC = 0.5

--------------------------------------
Samples mean AUC: 0.5000
======================================
```

3.6. Final Roc Auc Weighted

The Weighted ROC-AUC represents a balanced way to evaluate the overall performance of the model, mainly in unbalanced multiclass or multilabel contexts. Unlike the macro-average, which treats all classes equally, the weighted-average weights the AUC of each class by its support, that is, by the number of positive instances. In this way, more frequent classes influence the final result. It is especially useful when there are rare classes, as it prevents small variations in them from distorting the overall metric, maintaining a proportional and realistic evaluation.

```python
from sklearn import metrics
import numpy as np
import pandas as pd

def robust_weighted_roc_auc(true, pred, verbose=True):
    """
    Compute a robust weighted-average ROC-AUC with interpolation and special cases handling.

    ------------------------------------------------------------------------
    Description
    ------------------------------------------------------------------------
    This function computes the weighted-average ROC-AUC across multiple binary
    labels (multi-label classification). It is designed to handle special cases
    where certain labels have no class variation (all zeros or all ones),
    mixed binary predictions, or probabilistic predictions with constant truth.
    Each label's AUC is computed independently, and the final weighted-average
    score is obtained by weighting each label's AUC by its respective support
    (number of positive samples).

    ------------------------------------------------------------------------
    Objective
    ------------------------------------------------------------------------
    To provide a stable, interpretable, and statistically robust computation of
    weighted-average ROC-AUC scores, even when ROC curves are undefined for some
    labels. The weighting ensures that more frequent labels have proportionally
    greater influence on the final metric, making it suitable for imbalanced
    multi-label datasets.

    ------------------------------------------------------------------------
```

```
    Parameters
    ----------
    true : pandas.DataFrame
        Ground truth binary indicators (0 or 1) for each label column.

    pred : pandas.DataFrame
        Predicted scores or probabilities for each label column.
        Can contain either binary values (0/1) or continuous probabilities [0, 1].

    verbose : bool, default=True
        If True, prints detailed information for each label and the weighted results.

    ----------
    Returns
    ----------
    fpr_dict : dict
        A dictionary mapping each label to its computed False Positive Rates.

    tpr_dict : dict
        A dictionary mapping each label to its computed True Positive Rates.

    weighted_auc : float
        The weighted mean of individual label AUCs, where weights correspond
        to the relative support (number of positive samples) of each label.

    weighted_auc_df : pandas.DataFrame
        A DataFrame containing one row per label:
            - "Label": label name
            - "Support": number of positive samples
            - "Weight": proportion of total positives for that label
            - "AUC": computed AUC value
            - "FPR": array of false positive rates
            - "TPR": array of true positive rates

    ----------
    Example
    ----------
    >>> import pandas as pd
    >>> import numpy as np
    >>> from sklearn.datasets import make_multilabel_classification
    >>> X, Y = make_multilabel_classification(n_samples=100, n_features=10, n_classes=3)
    >>> true = pd.DataFrame(Y, columns=["Label1", "Label2", "Label3"])
    >>> pred = pd.DataFrame({
    ...     "Label1": np.random.rand(100),
    ...     "Label2": np.random.rand(100),
    ...     "Label3": np.random.rand(100)
    ... })
    >>> fpr_w, tpr_w, weighted_auc, weighted_auc_df = robust_weighted_roc_auc(true, pred)
```

```
    >>> print(weighted_auc_df)
           Label  Support  Weight    AUC
    0     Label1     51.0   0.480  0.701
    1     Label2     32.0   0.301  0.664
    2     Label3     23.0   0.217  0.629
    """

    aucs = []
    supports = []
    fpr_dict = {}
    tpr_dict = {}

    if verbose:
        print("\n" + "="*40)
        print("WEIGHTED-AVERAGE ROC")
        print("="*40)

    for col in true.columns:
        y_true = true[col].values
        y_pred = pred[col].values
        support = np.sum(y_true)
        supports.append(support)

        # --- CASE 1: y_true has only one class (no variation) ---
        if len(np.unique(y_true)) < 2:

            # (i) both empty (True=0, Pred=0)
            if np.all(y_true == 0) and np.all(y_pred == 0):
                auc = 1.0
                fpr_dict[col] = np.array([0.0, 1.0])
                tpr_dict[col] = np.array([0.0, 1.0])
                msg = f"{col}: True == 0 and PRED == 0 --> AUC = 1"

            # (ii) both full (True=1, Pred=1)
            elif np.all(y_true == 1) and np.all(y_pred == 1):
                auc = 1.0
                fpr_dict[col] = np.array([0.0, 1.0])
                tpr_dict[col] = np.array([0.0, 1.0])
                msg = f"{col}: True == 1 and PRED == 1 --> AUC = 1"

            # (iii) True empty, Pred full
            elif np.all(y_true == 0) and np.all(y_pred == 1):
                auc = 0.0
                fpr_dict[col] = np.array([0.0, 1.0])
                tpr_dict[col] = np.array([1.0, 1.0])
                msg = f"{col}: True == 0 and PRED == 1 --> AUC = 0"

            # (iv) True full, Pred empty
```

```
        elif np.all(y_true == 1) and np.all(y_pred == 0):
            auc = 0.0
            fpr_dict[col] = np.array([0.0, 1.0])
            tpr_dict[col] = np.array([0.0, 0.0])
            msg = f"{col}: True == 1 and PRED == 0 --> AUC = 0"

        # (v) True constant (0 or 1), but PRED has probabilities (not 0/1)
        elif len(np.unique(y_true)) == 1 and not np.all(np.isin(y_pred, [0, 1])):
            auc = 0.5
            fpr_dict[col] = np.array([0.0, 1.0])
            tpr_dict[col] = np.array([0.0, 1.0])
            true_val = int(y_true[0])
            msg = f"{col}: True == {true_val} and PRED probabilistic --> AUC = 0.5"

        # (vi) True all 1, Pred mixed (0/1)
        elif np.all(y_true == 1) and np.any(y_pred == 0) and np.any(y_pred == 1):
            auc = 1.0
            fpr_dict[col] = np.array([0.0, 1.0])
            tpr_dict[col] = np.array([0.0, 1.0])
            msg = f"{col}: True == 1 but PRED == 0/1 --> AUC = 1"

        # (vii) True all 0, Pred mixed (0/1)
        elif np.all(y_true == 0) and np.any(y_pred == 0) and np.any(y_pred == 1):
            auc = 0.0
            fpr_dict[col] = np.array([0.0, 1.0])
            tpr_dict[col] = np.array([0.0, 0.0])
            msg = f"{col}: True == 0 but PRED == 0/1 --> AUC = 0"

        # (viii) fallback
        else:
            auc = 0.5
            fpr_dict[col] = np.array([0.0, 1.0])
            tpr_dict[col] = np.array([0.0, 1.0])
            msg = f"{col}: Other special case --> AUC = 0.5"

    # --- CASE 2: normal case ---
    else:
        try:
            fpr, tpr, _ = metrics.roc_curve(y_true, y_pred)
            auc = metrics.auc(fpr, tpr)
            fpr_dict[col] = fpr
            tpr_dict[col] = tpr
            msg = f"{col}: Normal Case --> AUC={auc:.3f}"
        except ValueError:
            auc = 0.0
            fpr_dict[col] = np.array([0.0, 1.0])
            tpr_dict[col] = np.array([0.0, 0.0])
            msg = f"{col}: ROC computation failed --> AUC=0"
```

```
        aucs.append(auc)
        if verbose:
            print(msg)

    # --- weights computation ---
    supports = np.array(supports, dtype=float)
    if np.sum(supports) == 0:
        weights = np.ones_like(supports) / len(supports)
    else:
        weights = supports / np.sum(supports)

    auc_weighted = float(np.nansum(np.array(aucs) * weights))

    # --- detailed DataFrame ---
    auc_df_detailed = pd.DataFrame({
        "Label": true.columns,
        "Support": supports,
        "Weight": weights,
        "AUC": aucs,
        "FPR": [fpr_dict.get(col, np.array([])) for col in true.columns],
        "TPR": [tpr_dict.get(col, np.array([])) for col in true.columns],
    })

    if verbose:
        print("=" * 40)
        print(f"Weighted ROC AUC: {auc_weighted:.4f}")
        print("=" * 40)

    return fpr_dict, tpr_dict, auc_weighted, auc_df_detailed
```

Testing with a real true and pred labels with special cases:

```
true = pd.read_csv("~/AuprcRoc/data-real/GnegativeGO-results-lcc/Split-3/y_true.csv")
true
```

	Label1	Label2	Label3	Label4	Label5	Label6	Label7	Label8
0	1	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0
2	1	0	0	0	0	0	0	0
3	1	0	1	0	0	0	0	0
4	1	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...
135	0	0	0	0	0	0	0	1
136	0	0	0	0	0	0	0	1
137	0	0	0	0	0	0	0	1
138	0	0	0	0	0	0	0	1

	Label1	Label2	Label3	Label4	Label5	Label6	Label7	Label8
139	0	0	0	0	0	0	0	1

```
pred = pd.read_csv("~/AuprcRoc/data-real/GnegativeGO-results-lcc/Split-3/y_proba.csv")
pred
```

	Label1	Label2	Label3	Label4	Label5	Label6	Label7	Label8
0	1.000000	0.00	0.000	0.000	0.0	0.0	0.0	0.000
1	1.000000	0.00	0.000	0.000	0.0	0.0	0.0	0.000
2	1.000000	0.00	0.000	0.000	0.0	0.0	0.0	0.000
3	0.793333	0.01	0.500	0.000	0.0	0.0	0.0	0.005
4	0.960000	0.04	0.010	0.000	0.0	0.0	0.0	0.000
...	...	...	...	...	...	...	...	...
135	0.010000	0.00	0.000	0.015	0.0	0.0	0.0	0.935
136	0.000000	0.00	0.005	0.045	0.0	0.0	0.0	0.940
137	0.000000	0.00	0.000	0.000	0.0	0.0	0.0	1.000
138	0.000000	0.00	0.595	0.000	0.0	0.0	0.0	0.655
139	0.000000	0.00	0.000	0.000	0.0	0.0	0.0	0.985

```
roc_auc_score(true, pred)
```

```
/home/cissagatto/miniforge3/envs/ELCC/lib/python3.10/site-packages/sklearn/metrics/_ranking.py:424:
  warnings.warn(

nan
```

```
roc_auc_score(true, pred, average='weighted')
```

```
/home/cissagatto/miniforge3/envs/ELCC/lib/python3.10/site-packages/sklearn/metrics/_ranking.py:424:
  warnings.warn(

0.9985541865284753
```

```
fpr_dict, tpr_dict, auc_weighted, auc_df_detailed = robust_weighted_roc_auc(true, pred)
```

```
========================================
WEIGHTED-AVERAGE ROC
========================================
Label1: Normal Case --> AUC=0.999
Label2: Normal Case --> AUC=1.000
Label3: Normal Case --> AUC=0.998
Label4: Normal Case --> AUC=0.998
```

```
Label5: Normal Case --> AUC=0.996
Label6: Normal Case --> AUC=1.000
Label7: True == 0 and PRED probabilistic --> AUC = 0.5
Label8: Normal Case --> AUC=0.998
========================================
Weighted ROC AUC: 0.9986
========================================
```

```
fpr_dict
```

```
{'Label1': array([0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.01190476, 0.01190476, 0.03571429, 0.03571429, 0.10714286,
       0.13095238, 0.16666667, 0.21428571, 0.23809524, 0.3452381 ,
       0.48809524, 1.        ]),
 'Label2': array([0.       , 0.       , 0.       , 0.       , 0.0625   , 0.078125 ,
       0.1015625, 0.1171875, 0.1484375, 0.1953125, 0.234375 , 0.2421875,
       0.3046875, 1.       ]),
 'Label3': array([0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.01010101, 0.01010101, 0.02020202,
       0.02020202, 0.07070707, 0.09090909, 0.11111111, 0.13131313,
       0.16161616, 0.17171717, 0.21212121, 0.27272727, 0.29292929,
       0.33333333, 0.41414141, 1.        ]),
 'Label4': array([0.        , 0.        , 0.        , 0.        , 0.        ,
       0.00787402, 0.00787402, 0.01574803, 0.01574803, 0.07874016,
       0.09448819, 0.1023622 , 0.16535433, 0.19685039, 0.2519685 ,
       0.27559055, 0.37007874, 1.        ]),
 'Label5': array([0.        , 0.        , 0.        , 0.01470588, 0.01470588,
       0.05147059, 1.        ]),
 'Label6': array([0.        , 0.        , 0.02158273, 0.03597122, 1.        ]),
 'Label7': array([0., 1.]),
 'Label8': array([0.        , 0.        , 0.        , 0.        , 0.        ,
       0.00819672, 0.00819672, 0.02459016, 0.04098361, 0.07377049,
       0.09836066, 0.1147541 , 0.13114754, 0.13934426, 0.1557377 ,
       0.16393443, 0.18032787, 0.2295082 , 0.26229508, 0.30327869,
       0.36885246, 1.        ])}
```

```
tpr_dict
```

```
{'Label1': array([0.        , 0.48214286, 0.55357143, 0.57142857, 0.64285714,
       0.69642857, 0.80357143, 0.875     , 0.91071429, 0.96428571,
       0.96428571, 0.98214286, 0.98214286, 1.        , 1.        ,
       1.        , 1.        , 1.        , 1.        , 1.        ,
       1.        , 1.        ]),
 'Label2': array([0.  , 0.25, 0.5 , 1.  , 1.  , 1.  , 1.  , 1.  , 1.  , 1.  , 1.  ,
       1.  , 1.  , 1.  ]),
 'Label3': array([0.        , 0.29268293, 0.34146341, 0.53658537, 0.73170732,
```

```
       0.7804878 , 0.85365854, 0.85365854, 0.92682927, 0.92682927,
       1.        , 1.        , 1.        , 1.        , 1.        ,
       1.        , 1.        , 1.        , 1.        , 1.        ,
       1.        , 1.        , 1.        ]),
 'Label4': array([0.        , 0.30769231, 0.46153846, 0.61538462, 0.84615385,
       0.84615385, 0.92307692, 0.92307692, 1.        , 1.        ,
       1.        , 1.        , 1.        , 1.        , 1.        ,
       1.        , 1.        , 1.        ]),
 'Label5': array([0.  , 0.5 , 0.75, 0.75, 1.  , 1.  , 1.  ]),
 'Label6': array([0., 1., 1., 1., 1.]),
 'Label7': array([0., 1.]),
 'Label8': array([0.        , 0.11111111, 0.22222222, 0.33333333, 0.77777778,
       0.77777778, 1.        , 1.        , 1.        , 1.        ,
       1.        , 1.        , 1.        , 1.        , 1.        ,
       1.        , 1.        , 1.        , 1.        , 1.        ,
       1.        , 1.        ])}
```

```
auc_weighted
```

```
0.9985541865284753
```

```
auc_df_detailed
```

	Label	Support	Weight	AUC	FPR	TPR
0	Label1	56.0	0.386207	0.999150	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...	[0.0, 0.4821428571428
1	Label2	12.0	0.082759	1.000000	[0.0, 0.0, 0.0, 0.0, 0.0625, 0.078125, 0.10156...	[0.0, 0.25, 0.5, 1.0, 1.0
2	Label3	41.0	0.282759	0.997783	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.01010101...	[0.0, 0.2926829268292
3	Label4	13.0	0.089655	0.998183	[0.0, 0.0, 0.0, 0.0, 0.0, 0.007874015748031496...	[0.0, 0.3076923076923
4	Label5	4.0	0.027586	0.996324	[0.0, 0.0, 0.0, 0.014705882352941176, 0.014705...	[0.0, 0.5, 0.75, 0.75, 1.
5	Label6	1.0	0.006897	1.000000	[0.0, 0.0, 0.02158273381294964, 0.035971223021...	[0.0, 1.0, 1.0, 1.0, 1.0]
6	Label7	0.0	0.000000	0.500000	[0.0, 1.0]	[0.0, 1.0]
7	Label8	18.0	0.124138	0.998179	[0.0, 0.0, 0.0, 0.0, 0.0, 0.00819672131147541,...	[0.0, 0.1111111111111

Testing the new function with TRUE == 0 AND PRED == 0

```
true = pd.read_csv("~/AuprcRoc/data-fake/true_0.csv")
pred = pd.read_csv("~/AuprcRoc/data-fake/pred_0.csv")
fpr_dict, tpr_dict, auc_weighted, auc_df_detailed = robust_weighted_roc_auc(true, pred)
```

```
========================================
WEIGHTED-AVERAGE ROC
========================================
Label1: True == 0 and PRED == 0 --> AUC = 1
Label2: True == 0 and PRED == 0 --> AUC = 1
```

```
Label3: True == 0 and PRED == 0 --> AUC = 1
Label4: True == 0 and PRED == 0 --> AUC = 1
Label5: True == 0 and PRED == 0 --> AUC = 1
========================================
Weighted ROC AUC: 1.0000
========================================
```

Testing the new function with TRUE == 0 AND PRED == 1

```
true = pd.read_csv("~/AuprcRoc/data-fake/true_0.csv")
pred = pd.read_csv("~/AuprcRoc/data-fake/pred_1.csv")
fpr_dict, tpr_dict, auc_weighted, auc_df_detailed = robust_weighted_roc_auc(true, pred)
```

```
========================================
WEIGHTED-AVERAGE ROC
========================================
Label1: True == 0 and PRED == 1 --> AUC = 0
Label2: True == 0 and PRED == 1 --> AUC = 0
Label3: True == 0 and PRED == 1 --> AUC = 0
Label4: True == 0 and PRED == 1 --> AUC = 0
Label5: True == 0 and PRED == 1 --> AUC = 0
========================================
Weighted ROC AUC: 0.0000
========================================
```

Testing the new function with TRUE == 0 AND PRED == 1/0s

```
true = pd.read_csv("~/AuprcRoc/data-fake/true_0.csv")
pred = pd.read_csv("~/AuprcRoc/data-fake/pred_2.csv")
fpr_dict, tpr_dict, auc_weighted, auc_df_detailed = robust_weighted_roc_auc(true, pred)
```

```
========================================
WEIGHTED-AVERAGE ROC
========================================
Label1: True == 0 and PRED == 1 --> AUC = 0
Label2: True == 0 and PRED == 0 --> AUC = 1
Label3: True == 0 and PRED == 1 --> AUC = 0
Label4: True == 0 and PRED == 0 --> AUC = 1
Label5: True == 0 and PRED == 1 --> AUC = 0
========================================
Weighted ROC AUC: 0.4000
========================================
```

Testing the new function with TRUE == 0 AND PRED == 1/0s

```
true = pd.read_csv("~/AuprcRoc/data-fake/true_0.csv")
pred = pd.read_csv("~/AuprcRoc/data-fake/pred_3.csv")
fpr_dict, tpr_dict, auc_weighted, auc_df_detailed = robust_weighted_roc_auc(true, pred)
```

```
========================================
WEIGHTED-AVERAGE ROC
========================================
Label1: True == 0 but PRED == 0/1 --> AUC = 0
Label2: True == 0 but PRED == 0/1 --> AUC = 0
Label3: True == 0 but PRED == 0/1 --> AUC = 0
Label4: True == 0 but PRED == 0/1 --> AUC = 0
Label5: True == 0 but PRED == 0/1 --> AUC = 0
========================================
Weighted ROC AUC: 0.0000
========================================
```

Testing the new function with TRUE == 0 AND PRED == 1/0s

```
true = pd.read_csv("~/AuprcRoc/data-fake/true_0.csv")
pred = pd.read_csv("~/AuprcRoc/data-fake/pred_4.csv")
fpr_dict, tpr_dict, auc_weighted, auc_df_detailed = robust_weighted_roc_auc(true, pred)
```

```
========================================
WEIGHTED-AVERAGE ROC
========================================
Label1: True == 0 but PRED == 0/1 --> AUC = 0
Label2: True == 0 but PRED == 0/1 --> AUC = 0
Label3: True == 0 but PRED == 0/1 --> AUC = 0
Label4: True == 0 but PRED == 0/1 --> AUC = 0
Label5: True == 0 but PRED == 0/1 --> AUC = 0
========================================
Weighted ROC AUC: 0.0000
========================================
```

Testing the new function with TRUE == 0 AND PRED == probabilities

```
true = pd.read_csv("~/AuprcRoc/data-fake/true_0.csv")
pred = pd.read_csv("~/AuprcRoc/data-fake/pred_5.csv")
fpr_dict, tpr_dict, auc_weighted, auc_df_detailed = robust_weighted_roc_auc(true, pred)
```

```
========================================
WEIGHTED-AVERAGE ROC
========================================
```

```
Label1: True == 0 and PRED probabilistic --> AUC = 0.5
Label2: True == 0 and PRED probabilistic --> AUC = 0.5
Label3: True == 0 and PRED probabilistic --> AUC = 0.5
Label4: True == 0 and PRED probabilistic --> AUC = 0.5
Label5: True == 0 and PRED probabilistic --> AUC = 0.5
========================================
Weighted ROC AUC: 0.5000
========================================
```

Testing the new function with TRUE == 1 AND PRED == 1

```
true = pd.read_csv("~/AuprcRoc/data-fake/true_1.csv")
pred = pd.read_csv("~/AuprcRoc/data-fake/pred_1.csv")
fpr_dict, tpr_dict, auc_weighted, auc_df_detailed = robust_weighted_roc_auc(true, pred)
```

```
========================================
WEIGHTED-AVERAGE ROC
========================================
Label1: True == 1 and PRED == 1 --> AUC = 1
Label2: True == 1 and PRED == 1 --> AUC = 1
Label3: True == 1 and PRED == 1 --> AUC = 1
Label4: True == 1 and PRED == 1 --> AUC = 1
Label5: True == 1 and PRED == 1 --> AUC = 1
========================================
Weighted ROC AUC: 1.0000
========================================
```

Testing the new function with TRUE == 1 AND PRED == 0

```
true = pd.read_csv("~/AuprcRoc/data-fake/true_1.csv")
pred = pd.read_csv("~/AuprcRoc/data-fake/pred_0.csv")
fpr_dict, tpr_dict, auc_weighted, auc_df_detailed = robust_weighted_roc_auc(true, pred)
```

```
========================================
WEIGHTED-AVERAGE ROC
========================================
Label1: True == 1 and PRED == 0 --> AUC = 0
Label2: True == 1 and PRED == 0 --> AUC = 0
Label3: True == 1 and PRED == 0 --> AUC = 0
Label4: True == 1 and PRED == 0 --> AUC = 0
Label5: True == 1 and PRED == 0 --> AUC = 0
========================================
Weighted ROC AUC: 0.0000
========================================
```

Testing the new function with TRUE == 1 AND PRED == 0/1s

```
true = pd.read_csv("~/AuprcRoc/data-fake/true_1.csv")
pred = pd.read_csv("~/AuprcRoc/data-fake/pred_2.csv")
fpr_dict, tpr_dict, auc_weighted, auc_df_detailed = robust_weighted_roc_auc(true, pred)
```

```
========================================
WEIGHTED-AVERAGE ROC
========================================
Label1: True == 1 and PRED == 1 --> AUC = 1
Label2: True == 1 and PRED == 0 --> AUC = 0
Label3: True == 1 and PRED == 1 --> AUC = 1
Label4: True == 1 and PRED == 0 --> AUC = 0
Label5: True == 1 and PRED == 1 --> AUC = 1
========================================
Weighted ROC AUC: 0.6000
========================================
```

Testing the new function with TRUE == 1 AND PRED == 0/1s

```
true = pd.read_csv("~/AuprcRoc/data-fake/true_1.csv")
pred = pd.read_csv("~/AuprcRoc/data-fake/pred_3.csv")
fpr_dict, tpr_dict, auc_weighted, auc_df_detailed = robust_weighted_roc_auc(true, pred)
```

```
========================================
WEIGHTED-AVERAGE ROC
========================================
Label1: True == 1 but PRED == 0/1 --> AUC = 1
Label2: True == 1 but PRED == 0/1 --> AUC = 1
Label3: True == 1 but PRED == 0/1 --> AUC = 1
Label4: True == 1 but PRED == 0/1 --> AUC = 1
Label5: True == 1 but PRED == 0/1 --> AUC = 1
========================================
Weighted ROC AUC: 1.0000
========================================
```

Testing the new function with TRUE == 1 AND PRED == 0/1

```
true = pd.read_csv("~/AuprcRoc/data-fake/true_1.csv")
pred = pd.read_csv("~/AuprcRoc/data-fake/pred_4.csv")
fpr_dict, tpr_dict, auc_weighted, auc_df_detailed = robust_weighted_roc_auc(true, pred)
```

```
========================================
```

```
WEIGHTED-AVERAGE ROC
=======================================
Label1: True == 1 but PRED == 0/1 --> AUC = 1
Label2: True == 1 but PRED == 0/1 --> AUC = 1
Label3: True == 1 but PRED == 0/1 --> AUC = 1
Label4: True == 1 but PRED == 0/1 --> AUC = 1
Label5: True == 1 but PRED == 0/1 --> AUC = 1
=======================================
Weighted ROC AUC: 1.0000
=======================================
```

Testing the new function with TRUE == 1 AND PRED == probabilities

```
true = pd.read_csv("~/AuprcRoc/data-fake/true_1.csv")
pred = pd.read_csv("~/AuprcRoc/data-fake/pred_5.csv")
fpr_dict, tpr_dict, auc_weighted, auc_df_detailed = robust_weighted_roc_auc(true, pred)
```

```
=======================================
WEIGHTED-AVERAGE ROC
=======================================
Label1: True == 1 and PRED probabilistic --> AUC = 0.5
Label2: True == 1 and PRED probabilistic --> AUC = 0.5
Label3: True == 1 and PRED probabilistic --> AUC = 0.5
Label4: True == 1 and PRED probabilistic --> AUC = 0.5
Label5: True == 1 and PRED probabilistic --> AUC = 0.5
=======================================
Weighted ROC AUC: 0.5000
=======================================
```

Testing the new function with TRUE == 0/1 AND PRED == probabilities

```
true = pd.read_csv("~/AuprcRoc/data-fake/true_2.csv")
pred = pd.read_csv("~/AuprcRoc/data-fake/pred_5.csv")
fpr_dict, tpr_dict, auc_weighted, auc_df_detailed = robust_weighted_roc_auc(true, pred)
```

```
=======================================
WEIGHTED-AVERAGE ROC
=======================================
Label1: True == 1 and PRED probabilistic --> AUC = 0.5
Label2: True == 0 and PRED probabilistic --> AUC = 0.5
Label3: True == 1 and PRED probabilistic --> AUC = 0.5
Label4: True == 0 and PRED probabilistic --> AUC = 0.5
Label5: True == 1 and PRED probabilistic --> AUC = 0.5
=======================================
Weighted ROC AUC: 0.5000
=======================================
```

Testing the new function with TRUE == 0/1 AND PRED == probabilities

```
true = pd.read_csv("~/AuprcRoc/data-fake/true_3.csv")
pred = pd.read_csv("~/AuprcRoc/data-fake/pred_5.csv")
fpr_dict, tpr_dict, auc_weighted, auc_df_detailed = robust_weighted_roc_auc(true, pred)
```

```
========================================
WEIGHTED-AVERAGE ROC
========================================
Label1: Normal Case --> AUC=0.760
Label2: Normal Case --> AUC=0.600
Label3: Normal Case --> AUC=0.680
Label4: Normal Case --> AUC=0.760
Label5: Normal Case --> AUC=0.400
========================================
Weighted ROC AUC: 0.6400
========================================
```

4. AUPRC

Testing the new function with TRUE == 0 AND PRED == 0

```
true = pd.read_csv("~/AuprcRoc/data-fake/true_0.csv")
pred = pd.read_csv("~/AuprcRoc/data-fake/pred_0.csv")
```

Normal precision recall curve

```
precision, recall, _ = metrics.precision_recall_curve(y_true, y_pred)
```

```
precision
```

```
array([1., 1.])
```

```
recall
```

```
array([1., 0.])
```

Normal average precision recall curve

```
ap = metrics.average_precision_score(y_true, y_pred)
ap
```

```
1.0
```

Auprc Macro

```
from sklearn import metrics
import numpy as np
import pandas as pd

def robust_macro_auprc(true, pred, verbose=True):
    """
    Compute a robust macro-average Precision-Recall (AUPRC) with special cases handling.

    ------------------------------------------------------------------------
    Description
    ------------------------------------------------------------------------
    This function computes the macro-average Average Precision (AUPRC)
    across multiple binary labels (multi-label classification).
    It is designed to handle special cases where certain labels have no class
    variation (all zeros or all ones), mixed binary predictions, or
    probabilistic predictions with constant truth.
    Each label's AUPRC is computed independently, and the final macro-average
    score is obtained as the unweighted mean across all labels.

    ------------------------------------------------------------------------
    Objective
    ------------------------------------------------------------------------
    To provide a stable, interpretable, and statistically robust computation
    of macro-averaged AUPRC scores, even when the Precision-Recall curve
    cannot be defined by standard metrics. This makes it suitable for
    imbalanced multi-label datasets.

    ------------------------------------------------------------------------
    Parameters
    ------------------------------------------------------------------------
    true : pandas.DataFrame
        Ground truth binary indicators (0 or 1) for each label column.

    pred : pandas.DataFrame
        Predicted scores or probabilities for each label column.
        Can contain either binary values (0/1) or continuous probabilities [0, 1].

    verbose : bool, default=True
        If True, prints detailed information for each label and the macro results.

    ------------------------------------------------------------------------
    Returns
    ------------------------------------------------------------------------
    precision_dict : dict
        A dictionary mapping each label to its computed precision values.

    recall_dict : dict
        A dictionary mapping each label to its computed recall values.
```

```
    macro_ap : float
        The unweighted mean of individual label Average Precision scores.

    macro_ap_df : pandas.DataFrame
        A DataFrame containing one row per label:
            - "Label": label name
            - "AUPRC": computed Average Precision value
            - "Precision": array of precision values
            - "Recall": array of recall values

    ------------------------------------------------------------------------
    Example
    ------------------------------------------------------------------------
    >>> import pandas as pd
    >>> import numpy as np
    >>> from sklearn.datasets import make_multilabel_classification

    >>> X, Y = make_multilabel_classification(n_samples=100, n_features=10, n_classes=3)
    >>> true = pd.DataFrame(Y, columns=["Label1", "Label2", "Label3"])
    >>> pred = pd.DataFrame({
    ...     "Label1": np.random.rand(100),
    ...     "Label2": np.random.rand(100),
    ...     "Label3": np.random.rand(100)
    ... })

    >>> precision_dict, recall_dict, macro_ap, macro_ap_df = robust_macro_average_precision(true, pr
    >>> print(macro_ap_df)
          Label   AUPRC
    0    Label1  0.734
    1    Label2  0.681
    2    Label3  0.702
    """

    aps = []
    precision_dict = {}
    recall_dict = {}

    if verbose:
        print("\n" + "="*40)
        print("MACRO-AVERAGE PRECISION-RECALL (AUPRC)")
        print("="*40)

    for col in true.columns:
        y_true = true[col].values
        y_pred = pred[col].values

        # --- CASE 1: y_true has only one class (no variation) ---
        if len(np.unique(y_true)) < 2:
```

```
        # (i) both empty (True=0, Pred=0)
        if np.all(y_true == 0) and np.all(y_pred == 0):
            ap = 1.0
            precision_dict[col] = np.array([1.0])
            recall_dict[col] = np.array([0.0, 1.0])
            msg = f"{col}: True == 0 and PRED == 0 --> AP = 1"

        # (ii) both full (True=1, Pred=1)
        elif np.all(y_true == 1) and np.all(y_pred == 1):
            ap = 1.0
            precision_dict[col] = np.array([1.0])
            recall_dict[col] = np.array([0.0, 1.0])
            msg = f"{col}: True == 1 and PRED == 1 --> AP = 1"

        # (iii) True empty, Pred full
        elif np.all(y_true == 0) and np.all(y_pred == 1):
            ap = 0.0
            precision_dict[col] = np.array([0.0])
            recall_dict[col] = np.array([0.0, 1.0])
            msg = f"{col}: True == 0 and PRED == 1 --> AP = 0"

        # (iv) True full, Pred empty
        elif np.all(y_true == 1) and np.all(y_pred == 0):
            ap = 0.0
            precision_dict[col] = np.array([0.0])
            recall_dict[col] = np.array([0.0, 1.0])
            msg = f"{col}: True == 1 and PRED == 0 --> AP = 0"

        # (v) True constant, but PRED probabilistic
        elif len(np.unique(y_true)) == 1 and not np.all(np.isin(y_pred, [0, 1])):
            ap = 0.5
            precision_dict[col] = np.array([0.5])
            recall_dict[col] = np.array([0.0, 1.0])
            true_val = int(y_true[0])
            msg = f"{col}: True == {true_val} and PRED probabilistic --> AP = 0.5"

        # (vi) True all 1, Pred mixed (0/1)
        elif np.all(y_true == 1) and np.any(y_pred == 0) and np.any(y_pred == 1):
            ap = 1.0
            precision_dict[col] = np.array([1.0])
            recall_dict[col] = np.array([0.0, 1.0])
            msg = f"{col}: True == 1 but PRED == 0/1 --> AP = 1"

        # (vii) True all 0, Pred mixed (0/1)
        elif np.all(y_true == 0) and np.any(y_pred == 0) and np.any(y_pred == 1):
            ap = 0.0
            precision_dict[col] = np.array([0.0])
            recall_dict[col] = np.array([0.0, 1.0])
```

```
                msg = f"{col}: True == 0 but PRED == 0/1 --> AP = 0"

            # fallback
            else:
                ap = 0.5
                precision_dict[col] = np.array([0.5])
                recall_dict[col] = np.array([0.0, 1.0])
                msg = f"{col}: Other special case --> AP = 0.5"

        # --- CASE 2: normal ---
        else:
            try:
                precision, recall, _ = metrics.precision_recall_curve(y_true, y_pred)
                ap = metrics.average_precision_score(y_true, y_pred)
                precision_dict[col] = precision
                recall_dict[col] = recall
                msg = f"{col}: normal case → AP={ap:.3f}"
            except ValueError:
                ap = 0.0
                precision_dict[col] = np.array([0.0])
                recall_dict[col] = np.array([0.0, 1.0])
                msg = f"{col}: PR computation failed → AP=0"

        aps.append(ap)
        if verbose:
            print(msg)

    macro_ap = np.mean(aps)

    macro_ap_df = pd.DataFrame({
        "Label": true.columns,
        "AUPRC": aps,
        "Precision": [precision_dict[col] for col in true.columns],
        "Recall": [recall_dict[col] for col in true.columns],
    })

    if verbose:
        print("=" * 40)
        print(f"Macro Average Precision (AUPRC): {macro_ap:.4f}")
        print("=" * 40)

    return precision_dict, recall_dict, macro_ap, macro_ap_df
```

REFERENCES

Auc

Roc Curve

Roc Auc Score

Plot Roc

Precision Recall Curve

Average Precision Score

https://en.wikipedia.org/wiki/Confusion_matrix