

s09\_\_roc\_\_2

March 24, 2024

# 1 Mitsiu Alejandro Carreño Sarabia - E23S-18014

## 2 Introducción

El objetivo de este notebook es entrenar una red neuronal que pueda determinar una imagen de número manuscrito entre 0 y 9. Las características de las imágenes es que deben de ser de 8x8 pixeles (64 en total). Dónde cada pixel tiene un valor entre 0 y 16 (escala de grises).

Finalmente una vez obtenido el modelo de predicción se evaluará el rendimiento del clasificador mediante un analisis del área bajo la curva y las curvas ROC multiclase.

### 2.1 Carga de datos

```
[1]: from sklearn.datasets import load_digits

import numpy as np
import pandas as pd
```

```
[2]: data = load_digits()

print(data.DESCR)
print(data.feature_names)
```

```
.. _digits_dataset:
```

Optical recognition of handwritten digits dataset

-----

**\*\*Data Set Characteristics:\*\***

```
:Number of Instances: 1797
:Number of Attributes: 64
:Attribute Information: 8x8 image of integer pixels in the range 0..16.
:Missing Attribute Values: None
:Creator: E. Alpaydin (alpaydin '@' boun.edu.tr)
:Date: July; 1998
```

This is a copy of the test set of the UCI ML hand-written digits datasets

<https://archive.ics.uci.edu/ml/datasets/Optical+Recognition+of+Handwritten+Digits>

The data set contains images of hand-written digits: 10 classes where each class refers to a digit.

Preprocessing programs made available by NIST were used to extract normalized bitmaps of handwritten digits from a preprinted form. From a total of 43 people, 30 contributed to the training set and different 13 to the test set. 32x32 bitmaps are divided into nonoverlapping blocks of 4x4 and the number of on pixels are counted in each block. This generates an input matrix of 8x8 where each element is an integer in the range 0..16. This reduces dimensionality and gives invariance to small distortions.

For info on NIST preprocessing routines, see M. D. Garris, J. L. Blue, G. T. Candela, D. L. Dimmick, J. Geist, P. J. Grother, S. A. Janet, and C. L. Wilson, NIST Form-Based Handprint Recognition System, NISTIR 5469, 1994.

|details-start|  
\*\*References\*\*  
|details-split|

- C. Kaynak (1995) Methods of Combining Multiple Classifiers and Their Applications to Handwritten Digit Recognition, MSc Thesis, Institute of Graduate Studies in Science and Engineering, Bogazici University.
- E. Alpaydin, C. Kaynak (1998) Cascading Classifiers, Kybernetika.
- Ken Tang and Ponnuthurai N. Suganthan and Xi Yao and A. Kai Qin. Linear dimensionality reduction using relevance weighted LDA. School of Electrical and Electronic Engineering Nanyang Technological University. 2005.
- Claudio Gentile. A New Approximate Maximal Margin Classification Algorithm. NIPS. 2000.

|details-end|

['pixel\_0\_0', 'pixel\_0\_1', 'pixel\_0\_2', 'pixel\_0\_3', 'pixel\_0\_4', 'pixel\_0\_5', 'pixel\_0\_6', 'pixel\_0\_7', 'pixel\_1\_0', 'pixel\_1\_1', 'pixel\_1\_2', 'pixel\_1\_3', 'pixel\_1\_4', 'pixel\_1\_5', 'pixel\_1\_6', 'pixel\_1\_7', 'pixel\_2\_0', 'pixel\_2\_1', 'pixel\_2\_2', 'pixel\_2\_3', 'pixel\_2\_4', 'pixel\_2\_5', 'pixel\_2\_6', 'pixel\_2\_7', 'pixel\_3\_0', 'pixel\_3\_1', 'pixel\_3\_2', 'pixel\_3\_3', 'pixel\_3\_4', 'pixel\_3\_5', 'pixel\_3\_6', 'pixel\_3\_7', 'pixel\_4\_0', 'pixel\_4\_1', 'pixel\_4\_2', 'pixel\_4\_3', 'pixel\_4\_4', 'pixel\_4\_5', 'pixel\_4\_6', 'pixel\_4\_7', 'pixel\_5\_0', 'pixel\_5\_1', 'pixel\_5\_2', 'pixel\_5\_3', 'pixel\_5\_4', 'pixel\_5\_5', 'pixel\_5\_6', 'pixel\_5\_7', 'pixel\_6\_0', 'pixel\_6\_1', 'pixel\_6\_2', 'pixel\_6\_3', 'pixel\_6\_4', 'pixel\_6\_5', 'pixel\_6\_6', 'pixel\_6\_7', 'pixel\_7\_0', 'pixel\_7\_1', 'pixel\_7\_2', 'pixel\_7\_3', 'pixel\_7\_4', 'pixel\_7\_5', 'pixel\_7\_6', 'pixel\_7\_7']

```
[3]: X = pd.DataFrame(data.data, columns=data.feature_names)
      y = pd.DataFrame(data.target, columns=["Type"])
```

```
[4]: X.tail()
```

```
[4]:      pixel_0_0  pixel_0_1  pixel_0_2  pixel_0_3  pixel_0_4  pixel_0_5  \
1792         0.0         0.0         4.0         10.0         13.0         6.0
1793         0.0         0.0         6.0         16.0         13.0         11.0
1794         0.0         0.0         1.0         11.0         15.0         1.0
1795         0.0         0.0         2.0         10.0          7.0         0.0
1796         0.0         0.0        10.0         14.0          8.0         1.0

      pixel_0_6  pixel_0_7  pixel_1_0  pixel_1_1  ...  pixel_6_6  pixel_6_7  \
1792         0.0         0.0         0.0         1.0  ...         4.0         0.0
1793         1.0         0.0         0.0         0.0  ...         1.0         0.0
1794         0.0         0.0         0.0         0.0  ...         0.0         0.0
1795         0.0         0.0         0.0         0.0  ...         2.0         0.0
1796         0.0         0.0         0.0         2.0  ...         8.0         0.0

      pixel_7_0  pixel_7_1  pixel_7_2  pixel_7_3  pixel_7_4  pixel_7_5  \
1792         0.0         0.0         2.0        14.0        15.0         9.0
1793         0.0         0.0         6.0        16.0        14.0         6.0
1794         0.0         0.0         2.0         9.0        13.0         6.0
1795         0.0         0.0         5.0        12.0        16.0        12.0
1796         0.0         1.0         8.0        12.0        14.0        12.0

      pixel_7_6  pixel_7_7
1792         0.0         0.0
1793         0.0         0.0
1794         0.0         0.0
1795         0.0         0.0
1796         1.0         0.0
```

[5 rows x 64 columns]

```
[5]: y.tail()
```

```
[5]:      Type
1792     9
1793     0
1794     8
1795     9
1796     8
```

## 2.2 Estandarización

```
[6]: from sklearn.model_selection import train_test_split
     from sklearn.preprocessing import StandardScaler

[7]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
     ↪ random_state=117)

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

## 2.3 Implementar modelo

```
[8]: from tensorflow import keras
```

2024-03-24 05:05:05.474658: I tensorflow/core/util/port.cc:113] oneDNN custom operations are on. You may see slightly different numerical results due to floating-point round-off errors from different computation orders. To turn them off, set the environment variable `TF\_ENABLE\_ONEDNN\_OPTS=0`.

2024-03-24 05:05:05.538431: I tensorflow/core/platform/cpu\_feature\_guard.cc:210] This TensorFlow binary is optimized to use available CPU instructions in performance-critical operations.

To enable the following instructions: AVX2 AVX\_VNNI FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.

```
[9]: model = keras.Sequential([
     keras.layers.Input(shape=(64,)),
     #keras.layers.Dense(128, activation="relu"),
     keras.layers.Dense(12, activation="relu"),
     keras.layers.Dense(10, activation="softmax")
])

model.compile(optimizer="adam", loss="sparse_categorical_crossentropy",
     ↪ metrics=["accuracy"])
```

2024-03-24 05:05:07.434612: E external/local\_xla/xla/stream\_executor/cuda/cuda\_driver.cc:282] failed call to cuInit: UNKNOWN ERROR (34)

```
[10]: X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.
     ↪ 2, random_state=117)
```

## 2.4 Entrenamiento

Se aplicaron pocas épocas para disminuir la precisión del modelo

```
[11]: model.fit(X_train, y_train, epochs=2, validation_data=(X_val, y_val))
```

```
Epoch 1/2
36/36          1s 8ms/step -
accuracy: 0.1039 - loss: 2.6096 - val_accuracy: 0.1667 - val_loss: 2.3683
Epoch 2/2
36/36          0s 4ms/step -
accuracy: 0.2460 - loss: 2.0862 - val_accuracy: 0.3160 - val_loss: 1.9579
```

```
[11]: <keras.src.callbacks.history.History at 0x7fe637b85e50>
```

## 2.5 Evaluación de modelo

```
[12]: test_loss, test_accuracy = model.evaluate(X_test, y_test)
print(f"Perdida test: {test_loss:.4f}")
print(f"Precision test: {test_accuracy * 100 :.2f}%")
```

```
12/12          0s 2ms/step -
accuracy: 0.3589 - loss: 1.8285
Perdida test: 1.8492
Precision test: 34.72%
```

```
[13]: y_pred = model.predict(X_test)
```

```
12/12          0s 5ms/step
```

## 2.6 ROC

Las curvas ROC usualmente se componen de evaluar los verdaderos positivos (eje Y) contra 1 - los falsos positivos (eje X) siendo la esquina superior izq el punto ideal. También es posible calcular el área bajo la curva la cuál entre más se acerque a 1 es mejor.

## 2.7 One vs rest

Las curvas ROC usualmente se aplican a clasificaciones binarias, en los que los verdaderos positivos y falsos positivos son obvios. Cuando se trata de una categorización multiclase la noción de verdaderos positivos y falsos positivos se logra a traves de **binarizar los resultados**, esto se puede llevar a cabo de dos maneras.

- One vs Rest en el que se compara cada clase contra todas las demás (que se asumen como una sola)
- One vs One en el que se compara cada una de las combinaciones posibles de clases.

Durante este analisis únicamente se evaluó mediante el esquema One vs Rest en el que la clase en cuestion se interpreta como la clase positiva y el resto como la clase negativa

## 2.8 One-hot-encode OvR fashion

```
[14]: from sklearn.preprocessing import LabelBinarizer
```

```
[15]: label_binarizer = LabelBinarizer().fit(y_train)
      y_onehot_test = label_binarizer.transform(y_test)
```

```
[16]: print(y_onehot_test.shape)
      print(y_onehot_test[:10])
```

```
(360, 10)
[[0 0 0 0 0 0 1 0 0 0]
 [0 0 0 0 0 0 1 0 0 0]
 [1 0 0 0 0 0 0 0 0 0]
 [1 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 1 0 0 0 0 0]
 [0 0 0 0 0 0 0 1 0 0]
 [0 0 0 0 0 0 0 0 0 1]
 [0 0 0 1 0 0 0 0 0 0]
 [0 0 0 0 0 0 1 0 0 0]
 [1 0 0 0 0 0 0 0 0 0]]
```

```
[17]: for i in range(10):
      print(f"{i} == {label_binarizer.transform([i])}")
```

```
0 == [[1 0 0 0 0 0 0 0 0 0]]
1 == [[0 1 0 0 0 0 0 0 0 0]]
2 == [[0 0 1 0 0 0 0 0 0 0]]
3 == [[0 0 0 1 0 0 0 0 0 0]]
4 == [[0 0 0 0 1 0 0 0 0 0]]
5 == [[0 0 0 0 0 1 0 0 0 0]]
6 == [[0 0 0 0 0 0 1 0 0 0]]
7 == [[0 0 0 0 0 0 0 1 0 0]]
8 == [[0 0 0 0 0 0 0 0 1 0]]
9 == [[0 0 0 0 0 0 0 0 0 1]]
```

```
[18]: for i in range(10):
      class_id = np.flatnonzero(label_binarizer.classes_ == i)[0]
      print(f"{i} == {class_id}")
```

```
0 == 0
1 == 1
2 == 2
3 == 3
4 == 4
5 == 5
6 == 6
7 == 7
```

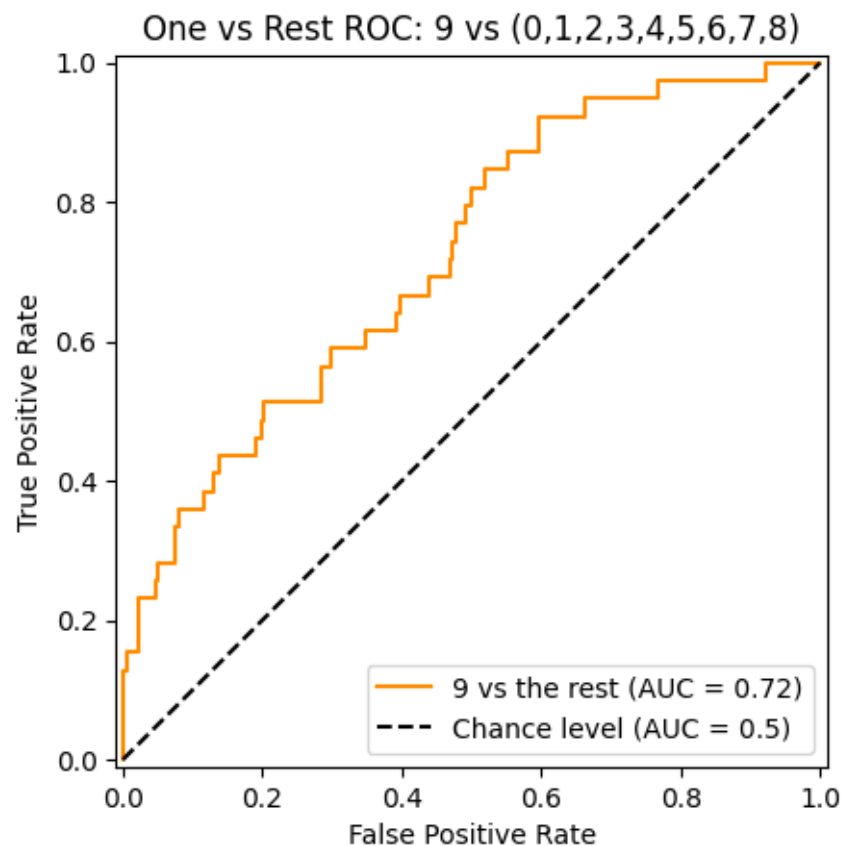
```
8 == 8
9 == 9
```

Nos damos cuenta que dada la naturaleza numérica de las clases **el onehot encoding genera exactamente los mismos resultados** que tomar la clase de manera directa, pero el procedimiento anterior es útil cuando tenemos categorías nominales.

```
[19]: import matplotlib.pyplot as plt

from sklearn.metrics import RocCurveDisplay

display = RocCurveDisplay.from_predictions(
    y_onehot_test[:, class_id],
    y_pred[:, i],
    name=f"{i} vs the rest",
    color="darkorange",
    plot_chance_level=True
)
_ = display.ax_.set(
    xlabel="False Positive Rate",
    ylabel="True Positive Rate",
    title="One vs Rest ROC: 9 vs (0,1,2,3,4,5,6,7,8)"
)
```



## 2.9 ROC con micro-promedio OvR

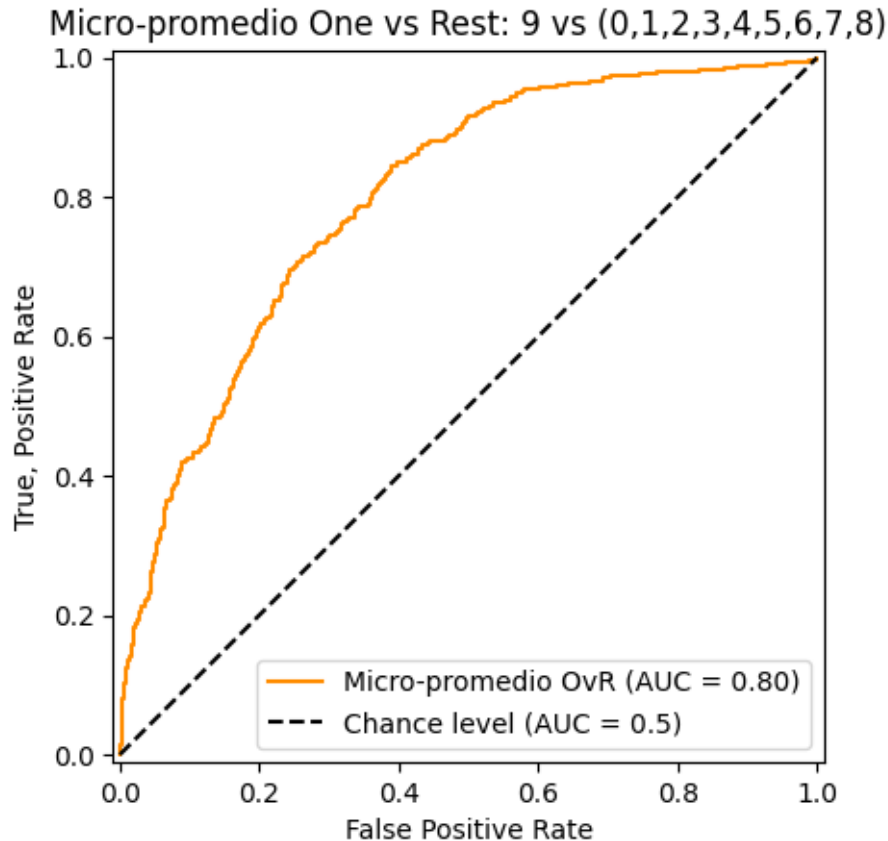
El micropromedio toma en cuenta las contribuciones de todas las clases. Es recomendable con clasificaciones muy imbalanceadas.

Para el micro-promedio se aplica el método `ravel` de `numpy` mediante el cuál se suman los `true positive`, `false positive`, `true negative` y `true positive` de cada una de las clases.

Ejemplo `y_pred = [ [1,2,3,4], [3,4,6,7] ]` `y_pred.ravel() == [1,2,3,4, 3,4,6,7]`

```
[20]: display = RocCurveDisplay.from_predictions(
    y_onehot_test.ravel(),
    y_pred.ravel(),
    name="Micro-promedio OvR",
    color="darkorange",
    plot_chance_level=True
)
_ = display.ax_.set(
    xlabel="False Positive Rate",
    ylabel="True, Positive Rate",
    title="Micro-promedio One vs Rest: 9 vs (0,1,2,3,4,5,6,7,8)"
)
```





Podemos notar que por tomar los promedios de cada categoría la curva se suaviza.

## 2.10 Calculando el micro-promedio de todas las categorías

Podemos observar que para obtener el micro-promedio es muy **similar a obtener el roc\_curve simple (una clasificación binaria) con la función roc\_curve(y\_real, y\_pred)**

También es posible obtener el valor del área con la función roc\_auc\_score en donde **se especifica el average micro y multi\_class ovr**

```
[21]: from sklearn.metrics import auc, roc_curve

fpr, tpr, roc_auc = dict(), dict(), dict()

# Micro-promedio
fpr["micro"], tpr["micro"], micro_thresholds = roc_curve(y_onehot_test.ravel(),
    ↪ y_pred.ravel())
roc_auc["micro"] = auc(fpr["micro"], tpr["micro"])

print(f"Micro-promedio One vs Rest ROC AUC: {roc_auc['micro']}")
```

```

#=====
from sklearn.metrics import roc_auc_score

micro_roc_auc_ovr = roc_auc_score(
    y_test,
    y_pred,
    multi_class="ovr",
    average="micro"
)
print(f"Micro-promedio función: {micro_roc_auc_ovr}")

```

Micro-promedio One vs Rest ROC AUC: 0.7989463305898491  
Micro-promedio función: 0.7989463305898491

```

[22]: fig, ax = plt.subplots(figsize=(10,10))

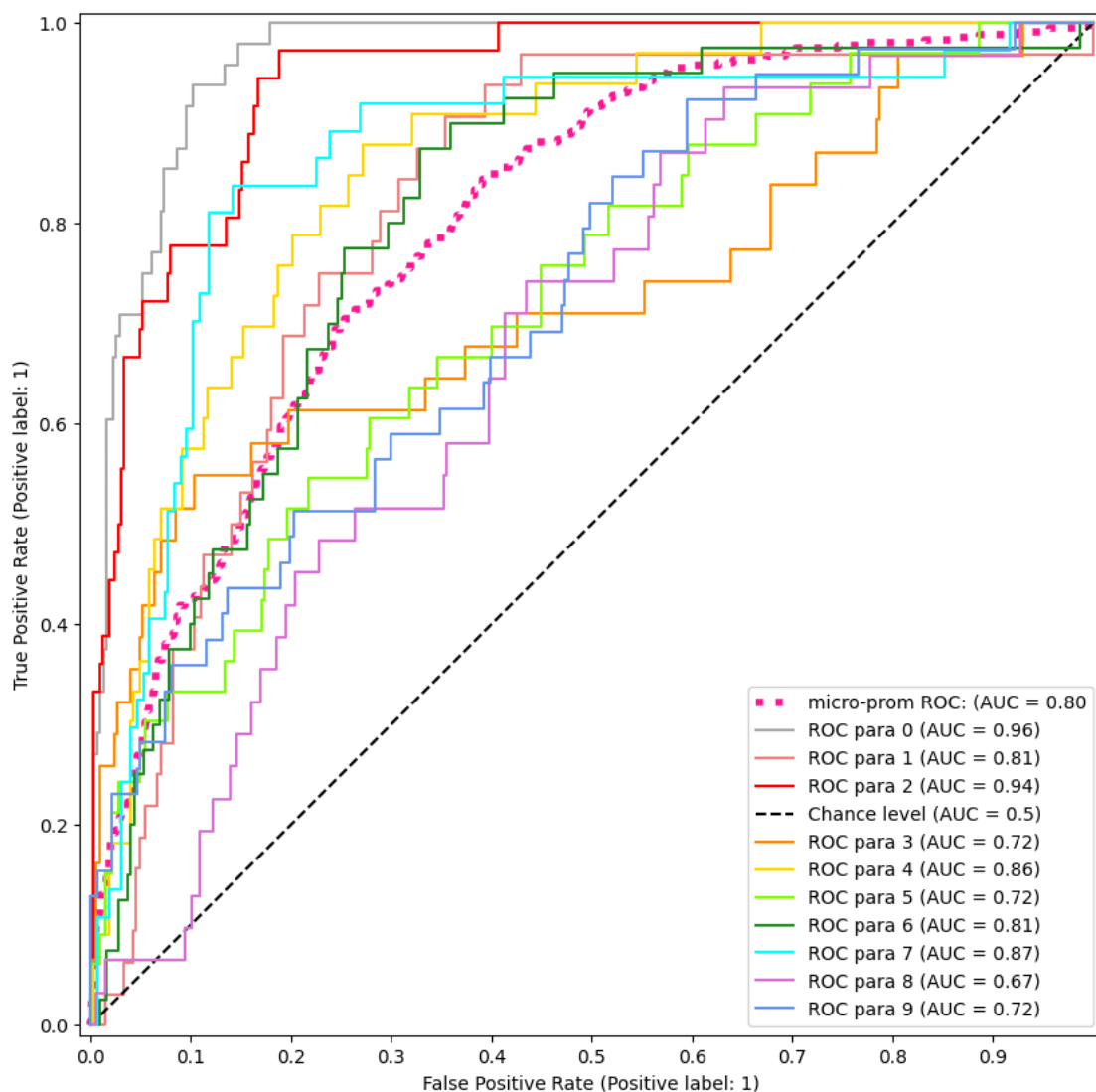
plt.xticks(np.arange(0, 1, 0.1))
# Dibujar el roc micro
plt.plot(
    fpr["micro"],
    tpr["micro"],
    label=f"micro-prom ROC: (AUC = {roc_auc['micro']:.2f})",
    color="deeppink",
    linestyle=":",
    linewidth=4
)

colors = ["darkgray",
          "lightcoral",
          "red",
          "darkorange",
          "gold",
          "chartreuse",
          "forestgreen",
          "aqua",
          "orchid",
          "cornflowerblue"]

# Dibujar cada una de las categorías
for i in range(10):
    RocCurveDisplay.from_predictions(
        y_onehot_test[:, i],
        y_pred[:, i],
        name=f"ROC para {i}",
        color=colors[i],
        ax = ax,
        plot_chance_level=(i == 2)
    )

```

)



## 2.11 ROC con macro-promedio OvR

```
[23]: for i in range(10):  
    fpr[i], tpr[i], _ = roc_curve(y_onehot_test[:, i], y_pred[:, i])  
    roc_auc[i] = auc(fpr[i], tpr[i])  
  
    # Start, end, num of elements  
    fpr_grid = np.linspace(0.0, 1.0, 1000)  
    #print(fpr_grid)  
  
    # Interpolar todas las curvas ROC en nuestro espacio
```

```

mean_tpr = np.zeros_like(fpr_grid)

for i in range(10):
    mean_tpr += np.interp(fpr_grid, fpr[i], tpr[i]) # interpolación lineal
    # Curve fitting using linear polynomials to construct new data points
    # within the range of a discrete set of known data points

mean_tpr /= 10

fpr["macro"] = fpr_grid
tpr["macro"] = mean_tpr
roc_auc["macro"] = auc(fpr["macro"], tpr["macro"])

print(f"Macro-promedio OvR ROC AUC: {roc_auc['macro']}")

#=====
macro_roc_auc_ovr = roc_auc_score(
    y_test,
    y_pred,
    multi_class="ovr",
    average="macro"
)
print(f"Macro-promedio función: {macro_roc_auc_ovr}")

```

Macro-promedio OvR ROC AUC: 0.808517605465743

Macro-promedio función: 0.8085802768768053

```

[24]: fig, ax = plt.subplots(figsize=(10,10))

plt.xticks(np.arange(0, 1, 0.1))
plt.plot(
    fpr["macro"],
    tpr["macro"],
    label=f"macro-prom ROC: (AUC = {roc_auc['macro']:.2f})",
    color="navy",
    linestyle=":",
    linewidth=4
)

colors = ["darkgray",
          "lightcoral",
          "red",
          "darkorange",
          "gold",
          "chartreuse",
          "forestgreen",
          "aqua",

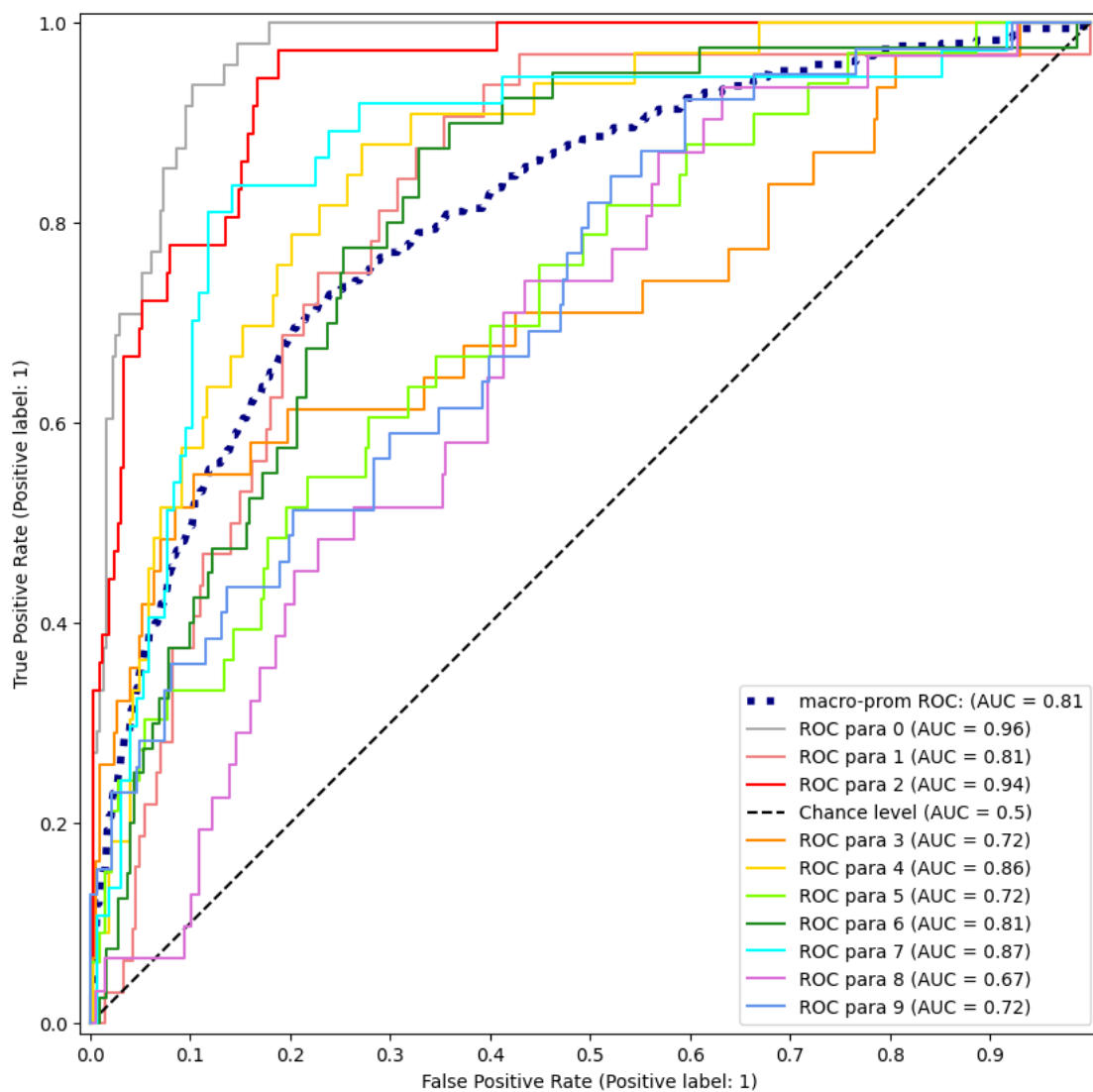
```

```

        "orchid",
        "cornflowerblue"]

for i in range(10):
    RocCurveDisplay.from_predictions(
        y_onehot_test[:, i],
        y_pred[:, i],
        name=f"ROC para {i}",
        color=colors[i],
        ax = ax,
        plot_chance_level=(i == 2)
    )

```



```

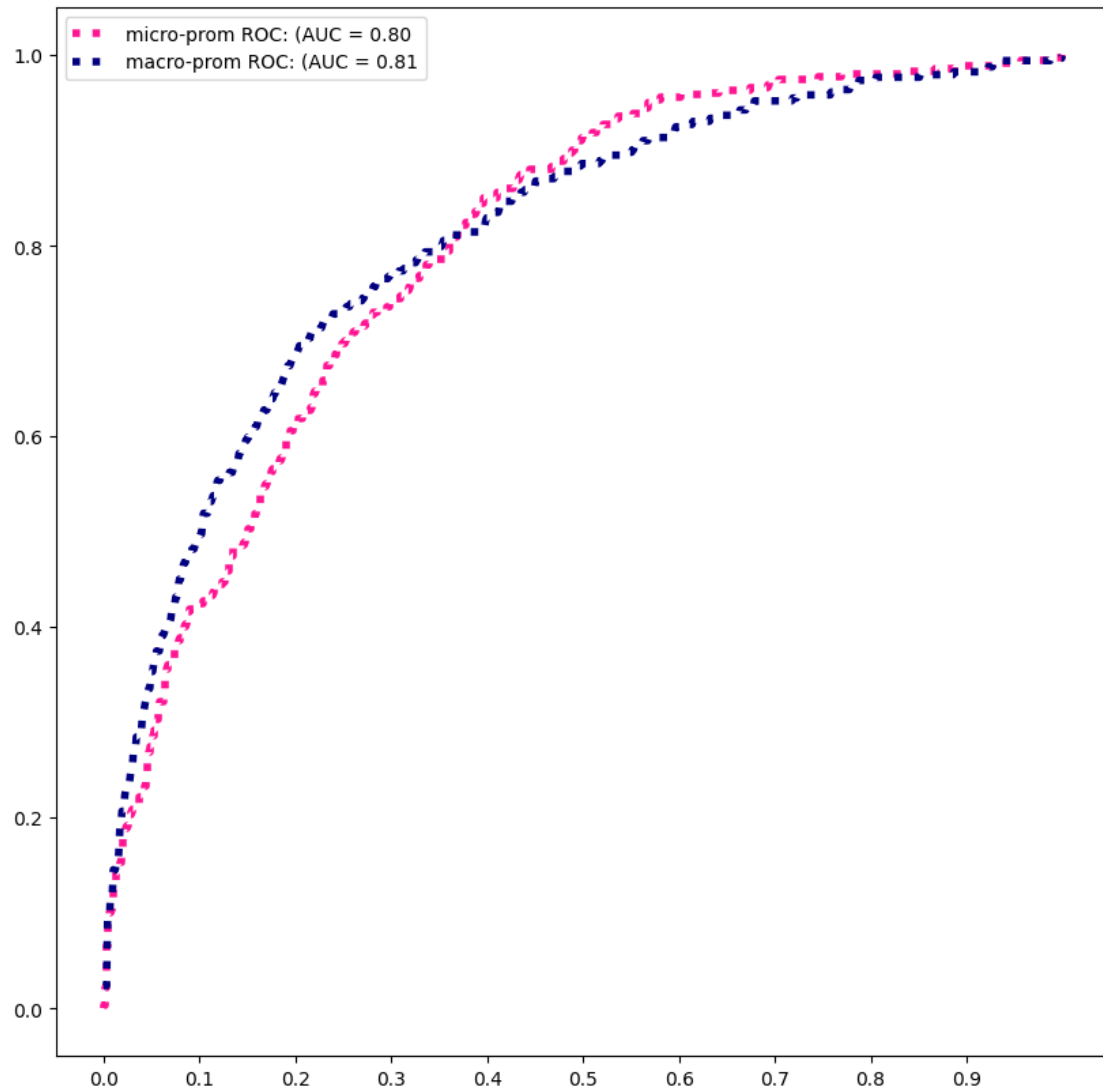
[25]: fig, ax = plt.subplots(figsize=(10,10))

plt.xticks(np.arange(0, 1, 0.1))
# Dibujar el roc micro
plt.plot(
    fpr["micro"],
    tpr["micro"],
    label=f"micro-prom ROC: (AUC = {roc_auc['micro']:.2f})",
    color="deeppink",
    linestyle=":",
    linewidth=4
)

plt.plot(
    fpr["macro"],
    tpr["macro"],
    label=f"macro-prom ROC: (AUC = {roc_auc['macro']:.2f})",
    color="navy",
    linestyle=":",
    linewidth=4
)

plt.legend(loc="best")
plt.show()

```



```
[26]: fig, ax = plt.subplots(figsize=(10,10))

plt.xticks(np.arange(0, 1, 0.1))
# Dibujar el roc micro
plt.plot(
    fpr["micro"],
    tpr["micro"],
    label=f"micro-prom ROC: (AUC = {roc_auc['micro']:.2f})",
    color="deeppink",
    linestyle=":",
    linewidth=4
)
```

```

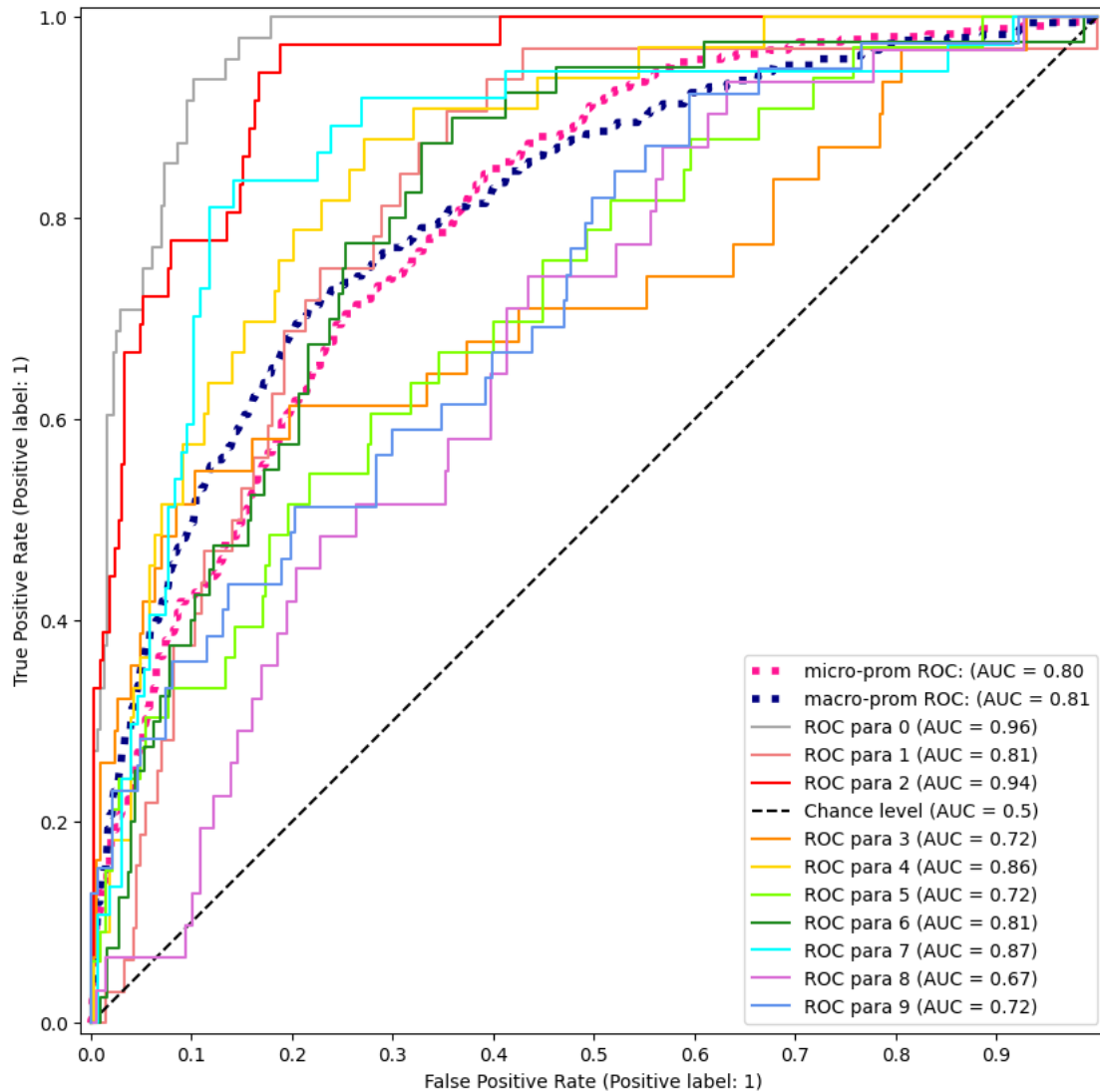
plt.plot(
    fpr["macro"],
    tpr["macro"],
    label=f"macro-prom ROC: (AUC = {roc_auc['macro']:.2f})",
    color="navy",
    linestyle=":",
    linewidth=4
)

colors = ["darkgray",
          "lightcoral",
          "red",
          "darkorange",
          "gold",
          "chartreuse",
          "forestgreen",
          "aqua",
          "orchid",
          "cornflowerblue"]

# Dibujar cada una de las categorías
for i in range(10):
    RocCurveDisplay.from_predictions(
        y_onehot_test[:, i],
        y_pred[:, i],
        name=f"ROC para {i}",
        color=colors[i],
        ax = ax,
        plot_chance_level=(i == 2)
    )

```





Conclusiones: Después de haber “empeorado” la red es mucho más sencillo notar cuales es la categoría que peor evalúa, y también se puede comparar con las otras categorías. Como interpretación personal puedo asumir que el dígito 8, así como el 9 son los peores clasificados dada su cercanía geométrica con el 0 (el mejor evaluado). (Los digitos cambiaron un poco al volver a ejecutar el notebook al agregar el punto óptimo de corte)

Creo que las funciones ROC multiclase es una herramienta poderosa que nos permite evaluar de manera aislada el desempeño de nuestros modelos de clasificación.

## 2.12 Calcular punto óptimo de corte

```
[27]: # Basado en índice de Youden
# tomando los true positive rate y false positive rate
youden = tpr["micro"] - fpr["micro"]
# Al ser un problema de optimización en la que queremos maximizar el true_
    ↳positive rate y minimizar el false positive rate
youden_argmax = np.argmax(youden)
# Buscamos en los thresholds que nos regresa ROC
cut_off = micro_thresholds[youden_argmax]

print(f"Punto de corte óptimo : {cut_off:.3f}")
```

Punto de corte óptimo : 0.092

```
[28]: print(micro_thresholds[np.argmax(tpr["micro"] - fpr["micro"])]))
```

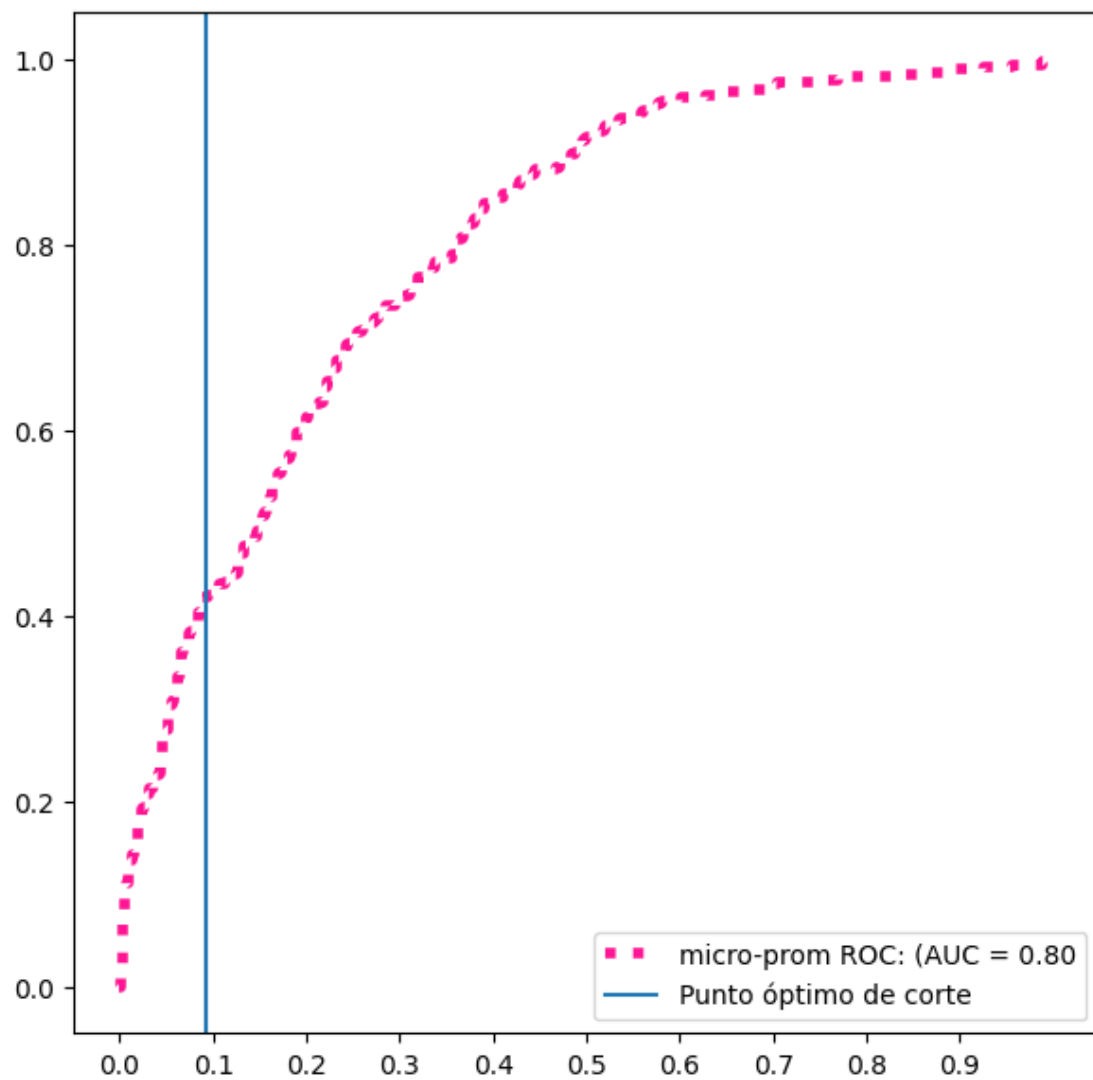
0.09171039

```
[29]: fig, ax = plt.subplots(figsize=(7,7))

plt.xticks(np.arange(0, 1, 0.1))
# Dibujar el roc micro
plt.plot(
    fpr["micro"],
    tpr["micro"],
    label=f"micro-prom ROC: (AUC = {roc_auc['micro']:.2f})",
    color="deeppink",
    linestyle=":",
    linewidth=4
)

plt.axvline(x=cut_off, label="Punto óptimo de corte")

plt.legend(loc="best")
plt.show()
```



[ ]: