Módulo 4. Métricas

CÓRDOBA CREA FUTURO

Introducción

La evaluación de modelos de inteligencia artificial requiere el uso de métricas que permitan medir de manera objetiva su rendimiento, precisión, eficiencia y fiabilidad. Estas métricas, tanto cuantitativas como cualitativas, posibilitan valorar la calidad de las predicciones, comparar distintos modelos y verificar que el sistema cumpla con los objetivos propuestos, no solo desde el punto de vista técnico, sino también estratégico y ético.

Según el tipo de tarea —clasificación, regresión, agrupamiento o generación de contenido— se emplean indicadores específicos que permiten analizar distintos aspectos del desempeño del modelo. Asimismo, en el caso de la IA generativa, se incorporan criterios relacionados con la calidad, la seguridad y la explicabilidad de los resultados.

A continuación, se presentan herramientas conceptuales y prácticas para interpretar, evaluar y mejorar modelos de inteligencia artificial, con énfasis en el análisis de métricas, la identificación de errores, los procesos de validación, la detección de sesgos y la toma de decisiones basada en datos.

¿Qué son las métricas en IA?

Las métricas en inteligencia artificial son indicadores numéricos que permiten evaluar el desempeño de un modelo. A través de ellas es posible determinar qué tan bien funciona un sistema, comparar distintos modelos, identificar errores y establecer si una solución resulta adecuada para un determinado contexto pedagógico o técnico. Asimismo, permiten analizar su impacto y los posibles riesgos asociados a su implementación.

Para comprender correctamente las métricas, es necesario conocer algunos conceptos básicos.

1 *Dataset*

Un dataset es el conjunto de datos utilizado en el proceso. Dentro de él, se distinguen los datos de entrenamiento, que el modelo emplea para aprender; los datos de validación, que se utilizan para ajustar parámetros; y los datos de prueba (*test*), que el modelo no ha visto previamente y sirven para evaluar su desempeño real.

2 **Etiquetas (*targets*)**

Las etiquetas o *targets* son las respuestas correctas con las que se comparan las predicciones del modelo. Por ejemplo, en un chatbot, las etiquetas pueden corresponder a intenciones como saludo, despedida o consulta_horario.

3 **Predicciones**

Las **predicciones** constituyen la salida generada por el modelo. Estas se comparan con las etiquetas reales para medir su rendimiento.

CONTINUAR

1. Métricas fundamentales en IA

Entre las métricas más utilizadas se encuentran las siguientes.

Accuracy (exactitud)

La *accuracy* o exactitud es la proporción de aciertos sobre el total de casos evaluados. Permite conocer qué porcentaje de las predicciones realizadas por el modelo coincide con las respuestas correctas.

Para calcularla, se divide la cantidad de aciertos obtenidos por el total de predicciones realizadas:

Accuracy=Aciertos/Total

Puede resultar engañosa cuando las clases están desbalanceadas. Por ejemplo, si el 90 % de los datos corresponde a la clase «saludo», un modelo que siempre prediga «saludo» alcanzará un 90 % de accuracy; sin embargo, se trataría de un modelo inadecuado.

Precision (precisión)

La precision o precisión mide, de todo lo que el modelo predijo como perteneciente a una clase, cuántos casos eran realmente correctos. Permite evaluar la confiabilidad de las predicciones y es especialmente útil cuando queremos minimizar las «falsas alarmas».

Se calcula dividiendo los verdaderos positivos entre la suma de verdaderos positivos y falsos positivos:

Precision = VerdaderosPositivos / VerdaderosPositivos + FalsosPositivos

Por ejemplo, en la detección de mensajes inapropiados, una alta precisión evita realizar acusaciones erróneas sobre contenidos válidos.

Recall (sensibilidad)

El *recall* o sensibilidad indica, de todos los casos que realmente pertenecían a una clase, cuántos fueron correctamente identificados por el modelo. Es útil cuando es crítico que el sistema no omita casos importantes.

Se calcula dividiendo los verdaderos positivos entre la suma de verdaderos positivos y falsos negativos:

Recall = VerdaderosPositivos / VerdaderosPositivos + FalsosNegativos

Por ejemplo, un modelo que detecta riesgo de abandono escolar debe tener un alto *recall* para identificar correctamente a los estudiantes en riesgo.

F1-*Score*

El F1-*Score* es el promedio armónico entre precisión (*precision*) y sensibilidad (*recall*). Permite medir el equilibrio entre ambos indicadores y se utiliza cuando se busca un desempeño general balanceado entre la detección correcta de positivos y la minimización de falsas alarmas.

F1 = 2·Precision·Recall / Precision + Recall

Es útil especialmente cuando las clases están equilibradas y se desea un equilibrio general en la evaluación del modelo.

Matriz de confusión

La **matriz de confusión** es una tabla que muestra la relación entre las predicciones del modelo y los valores reales:

Tabla 1. Matriz de confusión

Predicción \ Real	Clase A	Clase B
Predijo A	VP	FP

Predijo B	FN	VN

Fuente: elaboración propia

Permite analizar qué confunde el modelo y por qué, facilitando la identificación de errores y áreas de mejora en el desempeño.

Curvas ROC y AUC

Las **curvas ROC** y el **AUC** son métricas utilizadas principalmente en problemas de clasificación binaria. La curva ROC representa la relación entre la sensibilidad del modelo y la tasa de falsos positivos, mostrando su capacidad para distinguir correctamente entre clases.

El **AUC** (*area under the curve*) indica el área bajo la curva ROC. Un valor de 1,0 corresponde a un modelo perfecto, mientras que un valor de 0,5 indica un rendimiento equivalente al azar. Estas métricas resultan especialmente útiles para evaluar modelos predictivos y comparar su desempeño.

En el caso del **procesamiento del lenguaje natural (NLP)** y los bots, estas métricas permiten medir la precisión y la capacidad de los sistemas para clasificar intenciones, identificar entidades y responder correctamente ante distintos tipos de consultas.

La siguiente imagen resume todas las métricas comentadas y sirve como referencia rápida para evaluar algoritmos y modelos en proyectos de IA y NLP.

Figura 1. Resumen de métricas de evaluación en IA y NLP

Fuente: elaboración propia

Métricas específicas para NLP y bots

En el ámbito del procesamiento del lenguaje natural (NLP) y los bots, existen métricas particulares que permiten evaluar tanto la precisión en la comprensión de intenciones como la efectividad de la interacción conversacional.

- **Métricas por intención (*intent classification*)**
  Estas métricas se enfocan en medir cómo el modelo identifica correctamente la intención del usuario:
  - ***Accuracy* por intención.** Porcentaje de predicciones correctas para cada intención específica.
  - **Confusión entre intenciones:** analiza qué intenciones suelen confundirse entre sí.
  - **Tasa de «no entiendo» o «fallback»:** frecuencia con la que el bot no reconoce la intención y recurre a respuestas genéricas.
  - **Confianza promedio:** probabilidad asignada por el modelo a las predicciones realizadas.
  - **Entropía del modelo:** medida de incertidumbre en la clasificación, indicando qué tan seguro está el modelo de sus respuestas.

- **Métricas para bots conversacionales**
  Se dividen en objetivas y subjetivas:
  - **Objetivas**

- **Tasa de resolución:** proporción de mensajes resueltos sin intervención humana.
- **Interacciones por sesión:** número promedio de intercambios necesarios para completar una tarea.
- **Tiempo de respuesta:** rapidez con la que el bot contesta a cada mensaje.
- **Tasa de desvío:** porcentaje de casos en los que el bot requiere asistencia humana.

- **Subjetivas**
  - **Satisfacción del usuario:** evaluada mediante encuestas o calificaciones directas.
  - **Claridad del flujo conversacional:** percepción del usuario sobre lo entendible y coherente que resulta la interacción con el bot.

Métricas de calidad y ética en IA

En el contexto de la inteligencia artificial aplicada a la educación o a entornos institucionales, es fundamental evaluar no solo el desempeño técnico del modelo, sino también su equidad, seguridad y explicabilidad. Algunas métricas son las siguientes:

Sesgo (bias). —

Analiza si el modelo funciona de manera desigual con ciertos grupos de usuarios o si existen datos subrepresentados que podrían afectar la precisión de las predicciones.

Robustez: —

evalúa la capacidad del sistema para manejar entradas inesperadas, como errores de escritura, variaciones de jerga local o formulaciones inusuales de las consultas.

Interpretabilidad: —

determina si es posible explicar por qué el modelo falló o tomó una determinada decisión, facilitando la comprensión de su comportamiento.

Riesgo: —

mide el potencial del modelo para causar perjuicio, ya sea al usuario (por ejemplo, un estudiante) o al proceso institucional, asegurando que no se tomen decisiones incorrectas que afectan a las personas.

Recomendaciones de métricas

Para que tengas una guía sobre qué métricas usar según el tipo de problema, a continuación se presenta un cuadro resumido que te servirá al evaluar tus aplicaciones de IA y bots:

Tabla 2. Tipos de métricas

Problema	Métricas recomendadas	Justificación
Chatbot de intenciones	F1, matriz de confusión	Detectar confusiones comunes
Detección de riesgo escolar	*Recall*, AUC	Evitar falsos negativos
Clasificación de recursos educativos	*Precision*	Evitar sugerencias erróneas
Moderación automática	*Precision & Recall*	Equilibrar seguridad vs libertad

Fuente: elaboración propia

CONTINUAR

2. Métricas en un bot IA

A continuación se muestra un ejemplo práctico de cómo calcular métricas de evaluación como *precision*, *recall* y *F1* en un bot IA. Supongamos que el modelo clasifica mensajes de estudiantes en tres intenciones:

- Horario
- Entrega
- Saludo

Para comparar los resultados, utilizamos los siguientes comandos:

- `y_true` corresponde a la etiqueta real (verdad).
- `y_pred` corresponde a la predicción del modelo.

El siguiente código en Python calcula los verdaderos positivos (TP), falsos positivos (FP) y falsos negativos (FN) por clase, luego obtiene *precision*, *recall* y F1, y finalmente calcula el promedio macro de todas las clases. El código puede ejecutarse haciendo click en el siguiente link: https://www.programiz.com/online-compiler/1X1rpjLH5Qt1W

```
from collections import Counter

# Etiquetas reales (ground truth) y predichas por el modelo
y_true = [
                "saludo", "horario", "horario", "entrega", "entrega",
                "horario", "saludo", "entrega", "horario", "entrega",
                "saludo", "horario", "entrega", "horario", "entrega"
]

y_pred = [
                "saludo", "horario", "entrega", "entrega", "horario",
                "horario", "saludo", "entrega", "horario", "entrega",
                "horario", "horario", "entrega", "saludo", "entrega"
]

labels = sorted(set(y_true) | set(y_pred))
```

```
def tp_fp_fn(y_true, y_pred, etiqueta):
                """
                TP: predice etiqueta y era etiqueta
                FP: predice etiqueta y NO era etiqueta
                FN: NO predice etiqueta y ERA etiqueta
                """
                tp = sum((yt == etiqueta and yp == etiqueta) for yt, yp in zip(y_true, y_pred))
                fp = sum((yt != etiqueta and yp == etiqueta) for yt, yp in zip(y_true, y_pred))
                fn = sum((yt == etiqueta and yp != etiqueta) for yt, yp in zip(y_true, y_pred))
                return tp, fp, fn

def precision_recall_f1(tp, fp, fn):
                precision = tp / (tp + fp) if (tp + fp) else 0.0
                recall  = tp / (tp + fn) if (tp + fn) else 0.0
                f1              = 2*precision*recall/(precision+recall) if (precision+recall) else 0.0
                return precision, recall, f1

print("=== Métricas por clase (manual) ===")
metricas = {}

for lab in labels:
                tp, fp, fn = tp_fp_fn(y_true, y_pred, lab)
                p, r, f1 = precision_recall_f1(tp, fp, fn)
                metricas[lab] = (p, r, f1)
                print(f"\nClase: {lab}")
                print(f"TP={tp}, FP={fp}, FN={fn}")
                print(f"Precision={p:.2f} | Recall={r:.2f} | F1={f1:.2f}")

# Macro promedio (promedia métricas de clases)
macro_p = sum(metricas[l][0] for l in labels) / len(labels)
macro_r = sum(metricas[l][1] for l in labels) / len(labels)
macro_f = sum(metricas[l][2] for l in labels) / len(labels)

print("\n=== Promedios ===")
print(f"Macro Precision={macro_p:.2f}")
print(f"Macro Recall={macro_r:.2f}")
print(f"Macro F1={macro_f:.2f}")

# Para discusión: distribución de clases reales y predichas
print("\n=== Distribución ===")
print("Reales:", Counter(y_true))
print("Predichas:", Counter(y_pred))
```

Con este ejemplo podemos ver claramente cómo medir la efectividad del bot IA en la clasificación de intenciones, identificar errores comunes y comparar el desempeño de distintas configuraciones del modelo.

CONTINUAR

3. Interpretando las métricas

Matriz de confusión

Supongamos la matriz con los siguientes valores con ceros y unos:

Tabla 3. Ejemplo de matriz

Valor Real	Predicción
1	1
1	1
1	0
0	1
1	0
0	0

Fuente: elaboración propia

A partir de esta tabla, podemos contar los casos:

- **TP (*true positive*)**. Son los valores que el algoritmo clasifica como positivos y que realmente son positivos.
- **TN (*true negative*)**. Son valores que el algoritmo clasifica como negativos (0 en este caso) y que realmente son negativos.
- **FP (*false positive*)**. Falsos positivos, es decir, valores que el algoritmo clasifica como positivo cuando realmente son negativos.
- **FN (*false negative*)**. Falsos negativos, es decir, valores que el algoritmo clasifica como negativo cuando realmente son positivos.

La siguiente matriz se construye como resultado de observar los valores de la predicción con el valor real:

TP = 2. Las dos primeras filas tienen un 1 cada una indicando que se predijo que sería 1 y el valor real también fue 1.

Tabla 4. Ejemplo de verdaderos positivos (TP) en la matriz de confusión

Valor Real	Predicción
1	1
1	1

Fuente: elaboración propia

TN = 1. La última fila refleja esta condición

Tabla 5. Ejemplo de verdaderos negativos (TN) en la matriz de confusión

Valor Real	Predicción
0	0

Fuente: elaboración propia

FP = 1. Se encuentra un solo caso

Tabla 6. Ejemplo de falsos positivos (FP) en la matriz de confusión

Valor Real	Predicción
0	1

Fuente: elaboración propia

FN = 2. Se encuentran dos casos como los que se muestran

Tabla 7. Ejemplo de falsos negativos (FN) en la matriz de confusión

Valor Real	Predicción
1	0
1	0

Fuente: elaboración propia

Resumidamente, tenemos la siguiente matriz:

Tabla 8. Matriz resumida de la matriz de confusión

	1	0
1	TP = 2	FP = 1
0	FN = 2	TN = 1

Fuente: elaboración propia

La librería Scikit-Learn proporciona funciones para obtener una matriz de confusión de forma sencilla.

Figura 2. Ejemplo de código en Python para generar una matriz de confusión con Scikit-learn

```
from sklearn.metrics import confusion_matrix
y_true = [1, 1, 1, 0, 1, 0]
y_pred = [1, 1, 0, 1, 0, 0]
tn, fp, fn, tp = confusion_matrix(y_true, y_pred).ravel()
```

Fuente: captura de pantalla de librería Scikit-learn (https://scikit-learn.org/)

Accuracy

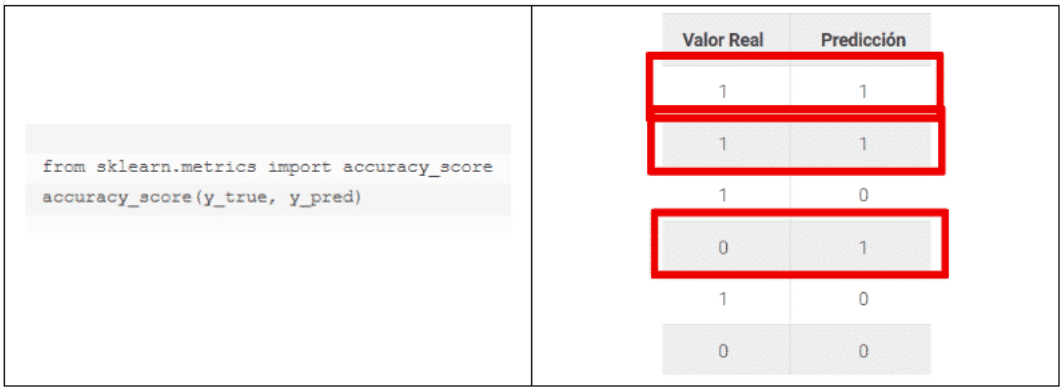
Representa el porcentaje de valores correctamente clasificados, tanto positivos como negativos. Se recomienda usarla cuando los datos están balanceados (igual cantidad de 0 y 1).

La fórmula es la siguiente:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

Fuente: elaboración propia

Figura 3. Cálculo de accuracy en Python y ejemplo de valores predichos vs. reales



```
from sklearn.metrics import accuracy_score
accuracy_score(y_true, y_pred)
```

Valor Real	Predicción
1	1
1	1
1	0
0	1
1	0
0	0

Fuente: elaboración propia

Del ejemplo anterior tendríamos un accuracy de 3/6, es decir, 50 %.

Precisión

Se utiliza para saber qué porcentaje de los valores que el modelo clasificó como positivos son realmente positivos. La fórmula es la siguiente:

$$\text{Precision} = \frac{TP}{TP + FP}$$

Fuente: elaboración propia

Figura 4. Cálculo de precision en Python y ejemplo de valores predichos vs. reales

```
from sklearn.metrics import precision_score
precision_score(y_true, y_pred)
```

Valor Real	Predicción
1	1
1	1
1	0
0	1
1	0
0	0

Fuente: elaboración propia

En el caso del ejemplo anterior, tendríamos una precision de 2/3, es decir, 66,6 %.

Recall

Se utiliza para saber cuántos de los valores positivos son correctamente identificados por el modelo. La fórmula es:

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

Fuente: elaboración propia

Figura 5. Cálculo de recall en Python y ejemplo de valores predichos vs. reales

```
from sklearn.metrics import recall_score
recall_score(y_true, y_pred)
```

Valor Real	Predicción
1	1
1	1
1	0
0	1
1	0
0	0

Fuente: elaboración propia

En el ejemplo anterior tendríamos un recall de 2/4, es decir, 50 %.

F1-Score

El F1-Score se utiliza en problemas donde el conjunto de datos está desbalanceado. Combina precision y recall para obtener un valor más representativo del desempeño del modelo. Su cálculo se expresa mediante la siguiente fórmula:

$$F1 = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

Fuente: elaboración propia

Figura 5. Cálculo de F1-Score en Python y ejemplo de valores predichos vs. reales

Fuente: elaboración propia

En el ejemplo, se obtiene un F1 de 2 × ((0,50 × 0,666) / (0,50 + 0,666)), lo que da como resultado 57,1 %.

Curva ROC (receiver operating characteristic)

La curva ROC es un gráfico que evalúa el rendimiento de un modelo de clasificación binaria. Muestra la tasa de verdaderos positivos (TPR) frente a la tasa de falsos positivos (FPR) para distintos umbrales de decisión.

¿Qué representa?

Observemos el siguiente gráfico:

Figura 3. Distribución de resultados según el punto de corte

Fuente: [imagen sin título sobre distribución de resultados según el punto de corte], (s.f.), https://goo.su/m4EU5N

Aquí, podemos observar los siguientes elementos:

EJE Y (ORDENADAS).	EJE X (ABSCISAS).
Corresponde a la tasa de verdaderos positivos (TPR), también denominada sensibilidad. Indica la proporción de casos positivos correctamente identificados.	

EJE Y (ORDENADAS).	EJE X (ABSCISAS).
	Corresponde a la tasa de falsos positivos (FPR), que equivale a 1 menos la especificidad. Representa la proporción de casos negativos clasificados incorrectamente como positivos.

¿Para qué sirve?

La curva ROC tiene diversas aplicaciones en la evaluación de modelos. Entre ellas, se encuentran las siguientes:

Evaluar el rendimiento de un modelo. —

Permite visualizar qué tan bien discrimina un clasificador entre dos grupos (por ejemplo, pacientes enfermos y sanos) a través de todos los umbrales posibles.

Seleccionar el punto de corte: —

ayuda a identificar el umbral más adecuado para una prueba o modelo, al buscar un equilibrio entre sensibilidad y especificidad.

Comparar modelos: —

facilita la comparación del rendimiento entre distintas pruebas diagnósticas o algoritmos de clasificación.

Calcular el área bajo la curva (AUC): —

el área bajo la curva ROC es una métrica que cuantifica la capacidad de un modelo para discriminar; cuanto más cercano a 1 sea el valor, mejor será su rendimiento.

Ejemplo de métricas con código Python

A continuación, se presenta un ejemplo de cálculo manual de precision, recall y F1 por clase, junto con el promedio macro y la distribución de etiquetas reales y predichas. El código define las funciones necesarias para obtener los verdaderos positivos (TP), falsos positivos (FP) y falsos negativos (FN), y luego calcula las métricas correspondientes.

```
from collections import Counter

# Etiquetas reales (ground truth) y predichas por el modelo

y_true = [

  "saludo", "horario", "horario", "entrega", "entrega",
```

```
  "horario", "saludo", "entrega", "horario", "entrega",
  "saludo", "horario", "entrega", "horario", "entrega"
]
y_pred = [
  "saludo", "horario", "entrega", "entrega", "horario",
  "horario", "saludo", "entrega", "horario", "entrega",
  "horario", "horario", "entrega", "saludo", "entrega"
]
labels = sorted(set(y_true) | set(y_pred))

def tp_fp_fn(y_true, y_pred, etiqueta):
  """
  TP: predice etiqueta y era etiqueta
  FP: predice etiqueta y NO era etiqueta
  FN: NO predice etiqueta y ERA etiqueta
  """
  tp = sum((yt == etiqueta and yp == etiqueta) for yt, yp in zip(y_true, y_pred))
  fp = sum((yt != etiqueta and yp == etiqueta) for yt, yp in zip(y_true, y_pred))
  fn = sum((yt == etiqueta and yp != etiqueta) for yt, yp in zip(y_true, y_pred))
  return tp, fp, fn
```

```
def precision_recall_f1(tp, fp, fn):
    precision = tp / (tp + fp) if (tp + fp) else 0.0
    recall    = tp / (tp + fn) if (tp + fn) else 0.0
    f1        = 2*precision*recall/(precision+recall) if (precision+recall) else 0.0
    return precision, recall, f1

print("=== Métricas por clase (manual) ===")
metricas = {}


for lab in labels:
    tp, fp, fn = tp_fp_fn(y_true, y_pred, lab)
    p, r, f1 = precision_recall_f1(tp, fp, fn)
    metricas[lab] = (p, r, f1)
    print(f"\nClase: {lab}")
    print(f"TP={tp}, FP={fp}, FN={fn}")
    print(f"Precision={p:.2f} | Recall={r:.2f} | F1={f1:.2f}")


# Macro promedio (promedia métricas de clases)
macro_p = sum(metricas[l][0] for l in labels) / len(labels)
macro_r = sum(metricas[l][1] for l in labels) / len(labels)
```

```
macro_f = sum(metricas[l][2] for l in labels) / len(labels)



print("\n=== Promedios ===")

print(f"Macro Precision={macro_p:.2f}")

print(f"Macro Recall={macro_r:.2f}")

print(f"Macro F1={macro_f:.2f}")

# Para discusión: distribución de clases reales y predichas

print("\n=== Distribución ===")

print("Reales:", Counter(y_true))

print("Predichas:", Counter(y_pred))
```

La ejecución del código produce la salida detallada por clase, seguida de los promedios macro y la distribución de frecuencias.

```
=== Métricas por clase (manual) ===

Clase: entrega

TP=5, FP=1, FN=1

Precision=0.83 | Recall=0.83 | F1=0.83

Clase: horario

TP=4, FP=2, FN=2

Precision=0.67 | Recall=0.67 | F1=0.67

Clase: saludo
```

TP=2, FP=1, FN=1

Precision=0.67 | Recall=0.67 | F1=0.67

=== Promedios ===

Macro Precision=0.72

Macro Recall=0.72

Macro F1=0.72

=== Distribución ===

Reales: Counter({'horario': 6, 'entrega': 6, 'saludo': 3})

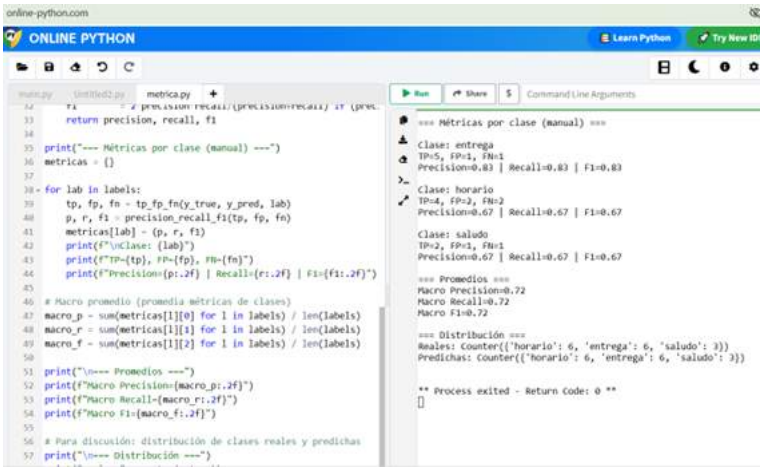
Predichas: Counter({'horario': 6, 'entrega': 6, 'saludo': 3})

En la siguiente imagen se muestra la ejecución del script en una IDE en línea, donde se visualizan las métricas por clase, los promedios macro y la distribución de etiquetas reales y predichas.

Figura 4. Ejecución de métricas por clase en IDE en línea

En la siguiente imagen se muestra la ejecución del script en una IDE en línea, donde se visualizan las métricas por clase, los promedios macro y la distribución de etiquetas reales y predichas.

Figura 4. Ejecución de métricas por clase en IDE en línea



Fuente: captura de pantalla de Online Python (https://www.online-python.com/YPQbfXi3t9)

CONTINUAR

Referencias

[Imagen sin título sobre distribución de resultados según el punto
https://www.sergas.gal/Docs/Profesional/PlataformaInnovacion/ItinerarioInvestigacion/material/PildoraFEGAS_Moni

Referencias bibliográficas de consulta

Bird, S., Klein, E., & Loper, E. (2009). *Natural Language Processing with Python*. O'Reilly Media. https://www.nltk.org

Jurafsky, D. & Martin, J. (2023). *Speech and Language Processing* (3rd ed. draft). MIT / Stanford. https://web.stanfor

Manning, C., Raghavan, P., & Schütze, H. (2008). *Introduction to Information Retrieval*. Cambr
https://nlp.stanford.edu/IR-book/

Ng, A. (2018). *Machine learning yearning*. Self-published.

Zheng, A. (2015). *Evaluating machine learning models: a beginner's guide to key concepts and pitfalls*. O'Reilly Med

CONTINUAR