Análisis de sentimientos en pueblos mexicanos

Descripción de código

Segundo modelo

De primera instancia, se tuvo que modificar el empleo de numpy, ya que, a la hora de realizar copias, por alguna razón, causaba problemas al ejecutar el código. Principalmente, en este segundo modelo, se hace uso de bert-base-spanish-wwm-cased, del usuario “dccuchile”.

1. Aquí, se instaura todo para iniciar a hacer el código. Se importan librerías, transformers, los archivos necesarios donde se encuentran los datos desde google drive. Se definen algunos nombres para usar estos archivos posteriormente.

```
import pandas as pd
from sklearn.model_selection import train_test_split
from transformers import AutoTokenizer, AutoModelForSequenceClassification, Trainer, TrainingArguments
from datasets import Dataset, DatasetDict
import torch
import numpy as np
import os # Importar la librería os para manejar rutas de archivos

# Definir la función de cálculo de métricas para evaluar el rendimiento del modelo durante el entrenamiento
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score
from google.colab import drive
drive.mount('/content/drive')

# --- 1. Cargar y preparar los datos ---

# Definir los nombres de los archivos CSV
LABELED_DATA_FILE = '/content/drive/MyDrive/MeIA_Reto/MeIA_2025_train.xlsx'
UNLABELED_DATA_FILE = '/content/drive/MyDrive/MeIA_Reto/MeIA_2025_test_wo_labels.xlsx'
OUTPUT_DATA_FILE = 'blog_data_con_sentimiento_predicho.csv'
MODEL_SAVE_PATH = "./fine_tuned_salida"
```

a. Se clasifican los datos y se convierten las clases a tipo entero.

```python
df_etiquetado = pd.read_excel(LABELED_DATA_FILE, sheet_name="MeIA_2025_train")
#Toma en cuenta que debes cambiar la ruta del archivo de tal modo que coincida con la ubicación
#de tu archivo
df_etiquetado['Polarity'] = df_etiquetado['Polarity'].astype(np.int64)#para convertir las clases a tipo entero

df_etiquetado = df_etiquetado.rename(columns={'Polarity': 'labels'})#para estandarizar con los modelos de transfomers
df_etiquetado['labels'] = df_etiquetado['labels'] - 1 #para que las clases estén de en el rango de [0-4]
df_etiquetado['labels'] = df_etiquetado['labels'].astype(int)
df_etiquetado = df_etiquetado.rename(columns={'Review': 'texto'})
df_etiquetado = df_etiquetado.drop(columns=['Town','Region','Type'])
#la escala del modelo val del 0 al 4 restamos uno por los datos que vienen
# en los datos del profe
df_etiquetado['labels'] = df_etiquetado['labels'].values.astype(np.int64)

#df_etiquetado['labels'] = df_etiquetado['labels'].sub(1)
df_etiquetado['labels'] = df_etiquetado['labels'].values.astype(np.int64)
df_etiquetado['labels'] = df_etiquetado['labels'].tolist()

print(f"Datos etiquetados")
print(type(df_etiquetado))
print(df_etiquetado)

#datos sin etiqueta

df_sin_etiquetar = pd.read_excel(UNLABELED_DATA_FILE, sheet_name="MeIA_2025_test_wo_labels")
df_sin_etiquetar= df_sin_etiquetar.drop(columns=['Town','Region','Type'])
#para que empate cuando se use el tokenizador
df_sin_etiquetar = df_sin_etiquetar.rename(columns={'Review': 'texto'})
#pf_test = pd.read_excel(UNLABELED_DATA_FILE, sheet_name="Hoja1")
print(f"Datos  SIN etiquetas")
print(type(df_sin_etiquetar))


df_sin_etiquetar.head()
```

2. Se perfila el código para emplear una función de tokenización para los datos.

```python
# Función de tokenización: convierte el texto en un formato numérico que el modelo entiende
def tokenize_function(examples):
    # 'truncation=True' corta los textos si son más largos que el máximo del modelo
    # 'padding="max_length"' rellena los textos cortos para que todos tengan la misma longitud
    # 'max_length=128' establece la longitud máxima de la secuencia (ajustable)
    return tokenizer(examples["texto"], truncation=True, padding="max_length", max_length=128)
```

3. En este bloque priorizamos la conversión de DataFrames a DataSets de HugginFace. De esta manera, el procesamiento del texto será óptimo.

```python
#a ver si jala, voy con la mano de Dios jajaja

# Dividir los datos etiquetados en conjuntos de entrenamiento y validación
# Usamos stratify para asegurar que la distribución de clases sea similar en ambos conjuntos
train_df, val_df = train_test_split(df_etiquetado, test_size=0.2, random_state=42, stratify=df_etiquetado['labels'])
# Convertir DataFrames a objetos Dataset de Hugging Face
train_dataset = Dataset.from_pandas(train_df, preserve_index=False)
val_dataset = Dataset.from_pandas(val_df, preserve_index=False)
unlabeled_dataset = Dataset.from_pandas(df_sin_etiquetar, preserve_index=False)

# Convertir DataFrames a objetos Dataset de Hugging Face
train_dataset = Dataset.from_pandas(train_df, preserve_index=False)
val_dataset = Dataset.from_pandas(val_df, preserve_index=False)
unlabeled_dataset = Dataset.from_pandas(df_sin_etiquetar, preserve_index=False)
```

a. En esta otra parte del código, se filtra el texto y se le aplica la función de tokenización a los datos, estableciendo así un nuevo formato de salida.

```python
# Asegúrate de que el número de etiquetas coincida con tu escala (0-4, es decir, 5 etiquetas)
model_name = "dccuchile/bert-base-spanish-wwm-cased"
num_labels = 5 # Correspondiente a la escala 0, 1, 2, 3, 4

print(f"\nCargando tokenizador y modelo pre-entrenado: {model_name}")
tokenizer = AutoTokenizer.from_pretrained(model_name)

# Aplicar la función de tokenización a todos los conjuntos de datos
print("Tokenizando los conjuntos de datos...")
tokenized_train_dataset = train_dataset.map(tokenize_function, batched=True)
tokenized_val_dataset = val_dataset.map(tokenize_function, batched=True)
tokenized_unlabeled_dataset = unlabeled_dataset.map(tokenize_function, batched=True)

# Eliminar las columnas originales de texto para dejar solo las necesarias para el modelo (input_ids, attention_mask, token_type_ids)
# y establecer el formato de salida como PyTorch tensors
tokenized_train_dataset.set_format("torch")
tokenized_val_dataset.set_format("torch")
tokenized_unlabeled_dataset.set_format("torch")

tokenized_train_dataset = tokenized_train_dataset.remove_columns(["texto"])
tokenized_val_dataset = tokenized_val_dataset.remove_columns(["texto"])
tokenized_unlabeled_dataset = tokenized_unlabeled_dataset.remove_columns(["texto"])
print("Tokenización completada.")



# Cargar el modelo pre-entrenado y configurarlo para la tarea de clasificación de secuencia
model = AutoModelForSequenceClassification.from_pretrained(model_name, num_labels=num_labels)
print("ok")
```

4. Establecemos todos los parámetros para dar inicio al entrenamiento del modelo

```python
# Configurar argumentos de entrenamiento:
# output_dir: Directorio donde se guardarán los checkpoints y el modelo final
# num_train_epochs: Número de veces que el modelo verá todo el dataset de entrenamiento
# per_device_train_batch_size: Número de ejemplos procesados por batch durante el entrenamiento
# per_device_eval_batch_size: Número de ejemplos procesados por batch durante la evaluación
# warmup_steps: Número de pasos para calentar la tasa de aprendizaje
# weight_decay: Regularización para evitar el sobreajuste
# logging_dir: Directorio para los logs (útil con TensorBoard)
# logging_steps: Frecuencia de logueo de métricas
# evaluation_strategy: Cuándo evaluar el modelo (ej. cada 'epoch')
# save_strategy: Cuándo guardar el modelo (ej. cada 'epoch')
# load_best_model_at_end: Cargar el mejor modelo (basado en 'metric_for_best_model') al final del entrenamiento
# metric_for_best_model: La métrica que se usará para determinar el 'mejor' modelo (eval_loss es común)
training_args = TrainingArguments(
    output_dir="./results",
    num_train_epochs=3, # Puedes ajustar este valor; 3-5 suelen ser un buen punto de partida
    per_device_train_batch_size=8, # Ajustar según tu memoria RAM/GPU (más pequeño si hay OutOfMemory)
    per_device_eval_batch_size=8,
    warmup_steps=500,
    weight_decay=0.01,
    logging_dir="./logs",
    logging_steps=10,
    eval_strategy="epoch",
    save_strategy="epoch", # Guardar el modelo al final de cada época
    load_best_model_at_end=True, # Cargar el mejor modelo guardado al finalizar el entrenamiento
    metric_for_best_model="eval_loss", # Utilizar la pérdida de validación para determinar el mejor modelo
    greater_is_better=False # Para eval_loss, un valor menor es mejor
)
```

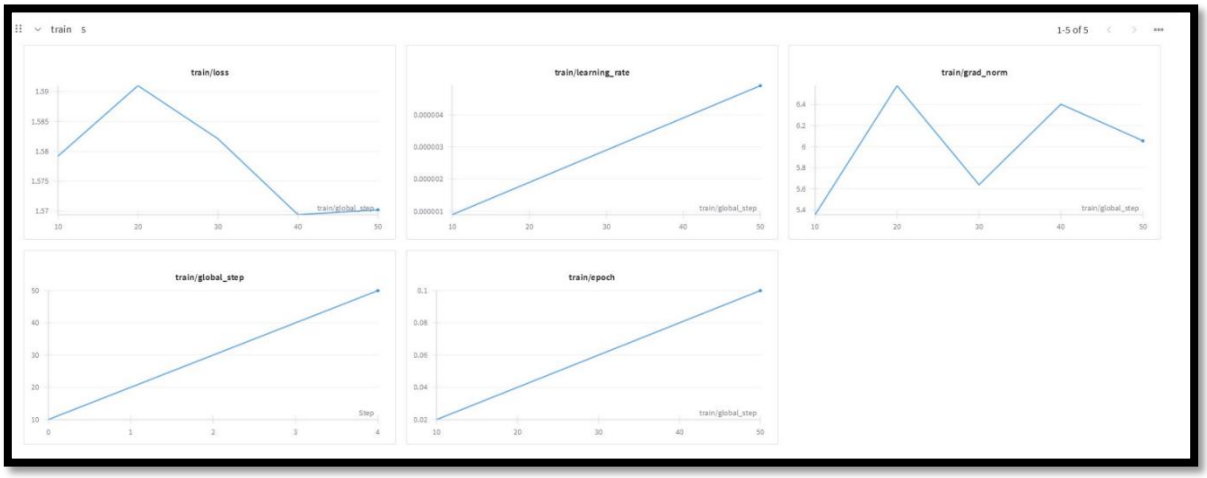
```python
    def compute_metrics(p):
        # p es una tupla (predictions, labels)
        predictions, labels = p
        # Convertir las logits (salidas crudas del modelo) a la etiqueta predicha (índice de la clase con mayor valor)
        predictions = np.argmax(predictions, axis=1)
        # Calcular métricas comunes
        accuracy = accuracy_score(labels, predictions)
        # f1_score 'weighted' es bueno para clases desbalanceadas, ya que considera el soporte de cada clase
        f1 = f1_score(labels, predictions, average='weighted')
        precision = precision_score(labels, predictions, average='weighted')
        recall = recall_score(labels, predictions, average='weighted')
        return {"accuracy": accuracy, "f1_score": f1, "precision": precision, "recall": recall}

    # Crear el objeto Trainer, que se encarga de la lógica de entrenamiento, evaluación y guardado

    trainer = Trainer(
        model=model,
        args=training_args,
        train_dataset=tokenized_train_dataset, # Pasa directamente el dataset de entrenamiento
        eval_dataset=tokenized_val_dataset,    # Pasa directamente el dataset de validación
        processing_class=tokenizer,
        compute_metrics=compute_metrics,
    )



    print("\n--- Iniciando el fine-tuning del modelo de sentimientos ---")
    trainer.train()
    print("Fine-tuning completado.")
```

5. Seguido de que afinamos el modelo a través de predicciones, generando datos ficticios. Así, el modelo entrenará y aprenderá a clasificar las reseñas.

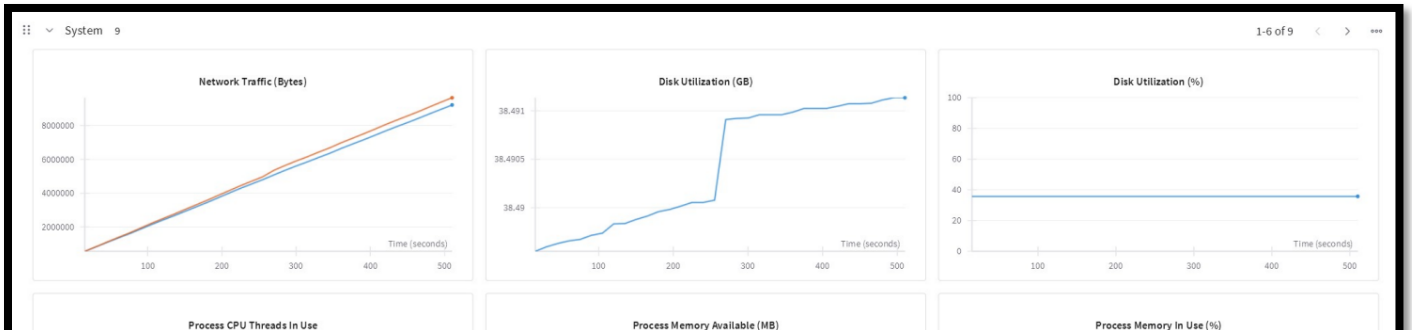
```python
#haciendo predicciones despues de tenerlo


predictor = Trainer(model=model, tokenizer=tokenizer)

print("\n--- Realizando predicciones en el conjunto de datos sin etiquetar ---")
# Pasa directamente el dataset sin etiquetar tokenizado al predictor
predictions_output = predictor.predict(tokenized_unlabeled_dataset)

# Las predicciones son logitos (valores sin escalar), necesitamos convertirlos a probabilidades
logits = predictions_output.predictions
# Convertir logits a probabilidades usando la función softmax
probabilities = torch.softmax(torch.tensor(logits), dim=1).numpy()

# Obtener la etiqueta predicha, que es el índice con la mayor probabilidad para cada texto
predicted_labels = np.argmax(probabilities, axis=1)

# Asignar las etiquetas predichas y las probabilidades al DataFrame original sin etiquetar
df_sin_etiquetar['sentimiento_predicho'] = predicted_labels
# Si quieres ver las probabilidades de cada clase, puedes guardar la lista completa
df_sin_etiquetar['probabilidades_clase'] = list(probabilities)

print("\n--- Vista previa de los resultados de las predicciones ---")
print(df_sin_etiquetar.head())
```

A continuación, se adjuntan algunas gráficas de la evolución del primer entrenamiento del modelo.

Entrenamiento 1:

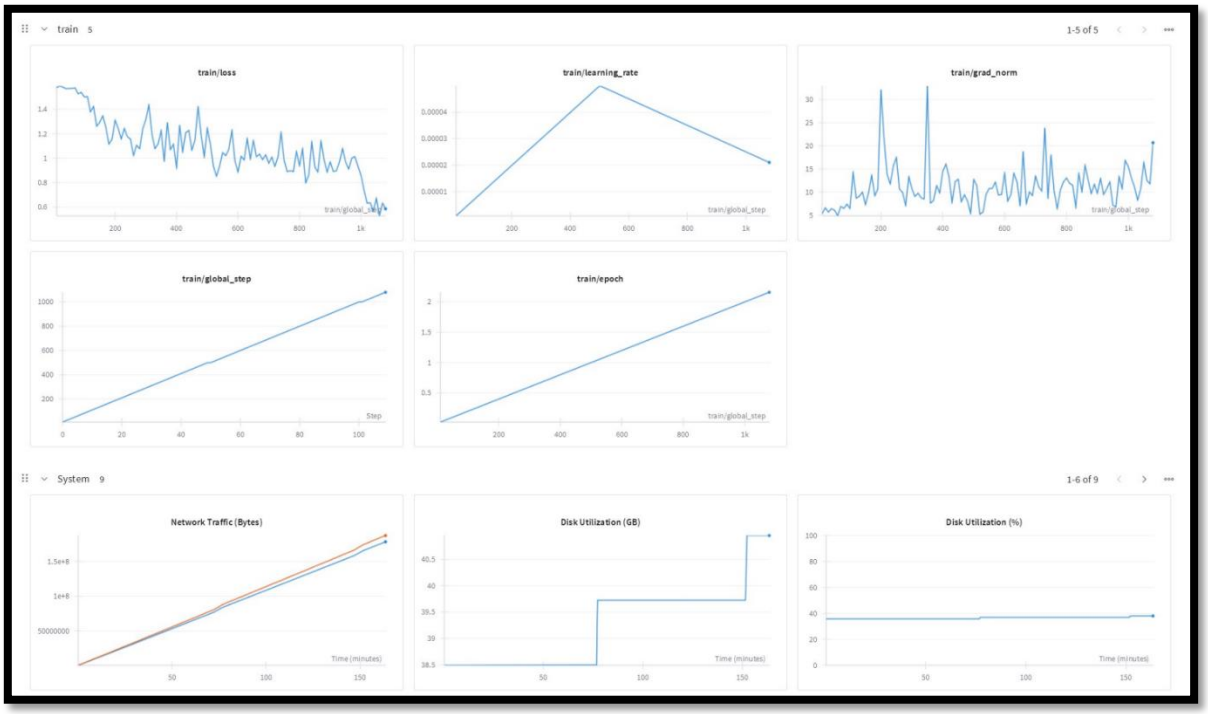


Rendimiento 1:

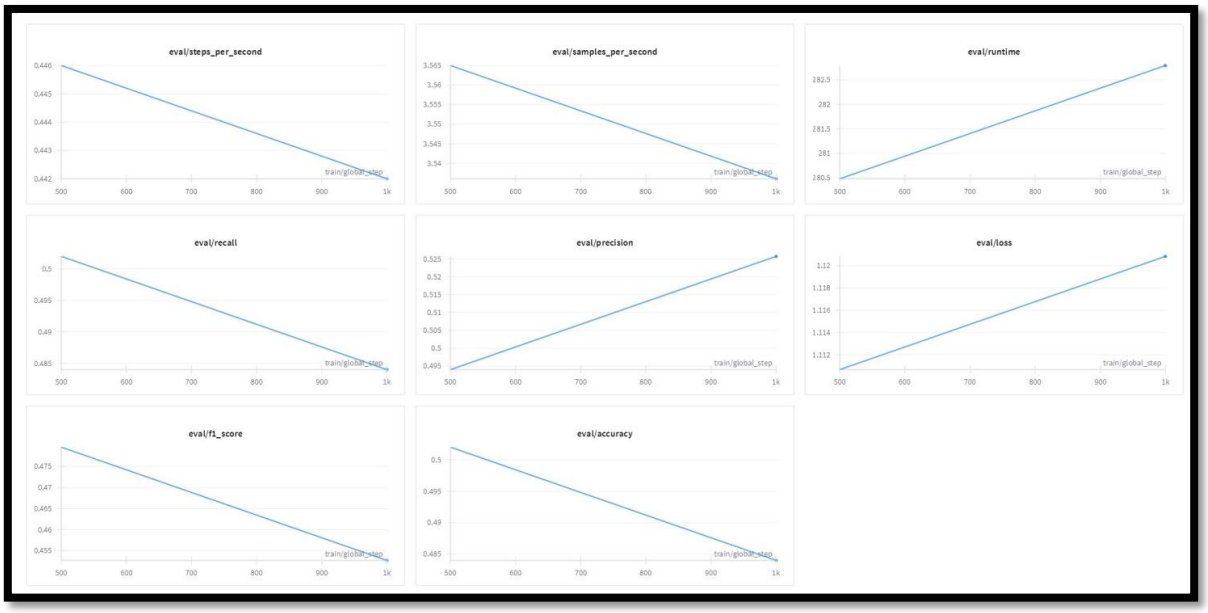


Ahora, se presentan las gráficas de otro entrenamiento, así como el rendimiento del equipo y la evaluación del modelo.

Entrenamiento y rendimiento 2:

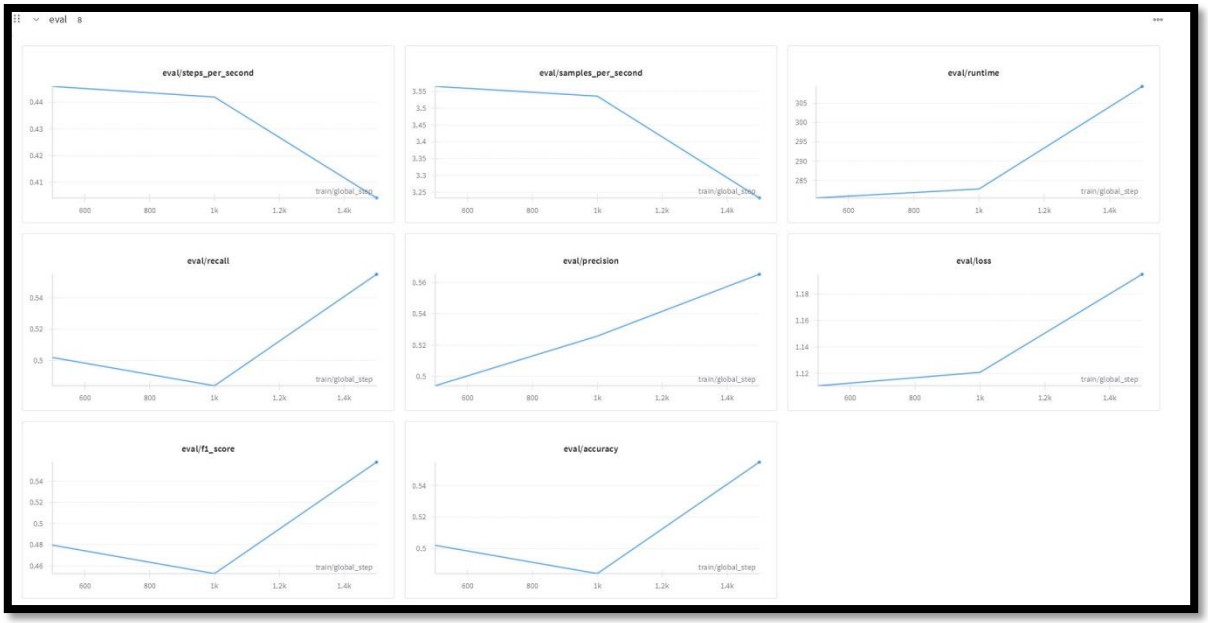


Evaluación 2:

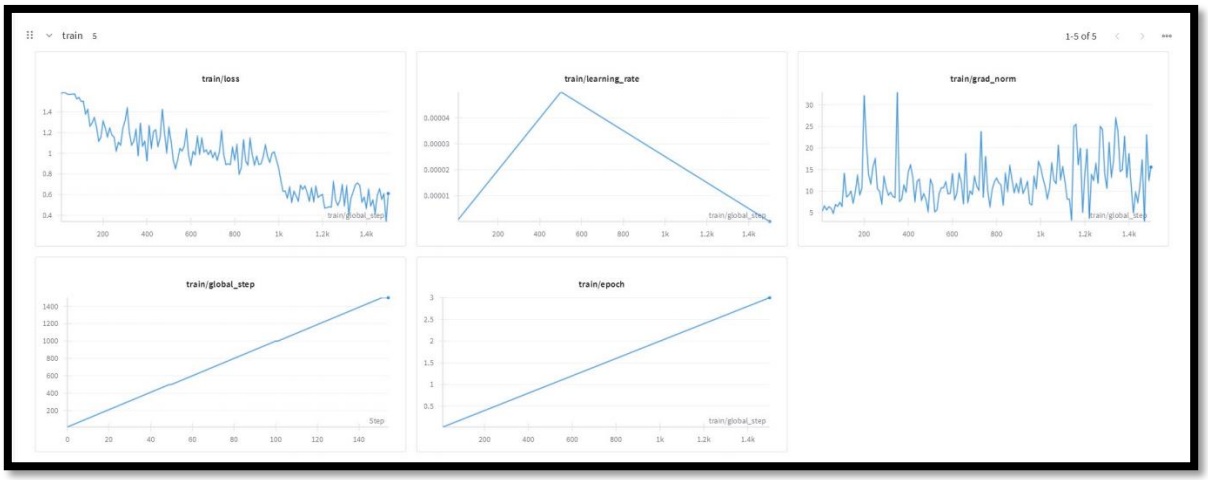


Mientras que, en una tercera evaluación, obtuvimos los siguientes gráficos.

Evaluación 3:



Entrenamiento 3:



Rendimiento 3: