Evidencia 1: Implementación de técnicas de aprendizaje máquina. (Portafolio Implementación)

Guadalupe Paulina López Cuevas A01701095@tec.mx

Resumen—

En este proyecto tiene la implementación de un sistema de detección de noticias falsas basado en técnicas de deep learning. El proyecto se enfoca en la preparación del dataset, el preprocesamiento y el desarrollo de un modelo inicial de clasificación utilizando arquitecturas neuronales recurrentes. El flujo de trabajo está estructurado en tres notebooks: ETL, entrenamiento y predicción, para garantizar modularidad y reproducibilidad. El dataset, obtenido de la plataforma Hugging Face, consiste en artículos etiquetados como noticias reales y falsas. Después de una limpieza exhaustiva, transformación, vectorización y generación del vocabulario, los datos se dividieron en conjuntos balanceados de train, validation y test. Se entrenó un modelo LSTM bidireccional con capas de regularización, alcanzando una precisión estable y demostrando ausencia de sobreajuste durante las primeras pruebas.

I. Introducción

La detección automática de noticias falsas se ha convertido en un desafío relevante dentro del procesamiento de lenguaje natural (NLP) debido al impacto social que la desinformación genera en ámbitos políticos, económicos y culturales. Debido a esto, este proyecto desarrolla un sistema de clasificación binaria enfocado en distinguir entre noticias reales y falsas mediante técnicas basadas en deep learning.

Para este proyecto, se utilizó el dataset "Fake News Detection" de Hugging Face, el cual incluye textos categorizados en dos clases. Este proyecto está organizado en tres notebooks: ETL (extracción, transformación y limpieza), ENT (entrenamiento y validación del modelo) y PRED (predicciones en datos nuevos). Esta estructura permite un flujo de trabajo modular y escalable.

Durante la fase ETL se preparó el dataset mediante limpieza, eliminación de columnas irrelevantes, verificación de valores nulos, reordenamiento (shuffle) de observaciones y vectorización de texto utilizando TextVectorization de Keras. Después se definió la arquitectura inicial basada en una capa Embedding, redes LSTM bidireccionales y capas densas con regularización mediante Dropout.

Los resultados mostraron un desempeño consistente entre las métricas de entrenamiento y validación, lo que indico buena capacidad de generalización. Esta entrega constituye la base para implementar mejoras al modelo en etapas posteriores, con el objetivo de optimizar precisión, estabilidad y robustez ante texto real no estructurado.

II. Información sobre dataset

El dataset "Fake News Detection" es un dataset obtenido a partir de la plataforma Hugging Face en el que se encuentra una recopilación de noticias reales y falsas (en inglés).

Este dataset cuenta con 6 atributos:

Unnamed	int	ID de identificación
title	String	Título de la noticia
text	String	Párrafo
subject (sin unidad)	String	Categoría de la noticia
date	Datetime	Fecha en la que se publicó la noticia
label	Booleano	Fake news = 0 True news = 1

II. Especificaciones sobre el proyecto

Para este proyecto se trabajará con Notebook Colab, en este caso el proyecto estará seccionado en 3 etapas que se encontraran de igual forma divididas en 3 notebooks, que se conectaran a través de exportación e importación de archivos por medio de Google drive:

- Notebook 1 (ETL): El objetivo de este archivo es importar el dataset original, hacer limpieza y transformaciones necesarias para preparar los datos para el modelo, crear el vocabulario para el modelo y hacer la separación en train, validation y test datasets para entrenar, validar y probar el modelo.
- Notebook 2 (ENT): El objetivo de este archivo es la definición de la arquitectura del modelo, entrenamiento del modelo y validación del modelo.
- Notebook 3 (PRED): El objetivo de este archivo es importar el modelo ya entrenado y hacer predicciones con datos nuevos, además de tener la opción de introducir texto nuevo para generar predicciones.

IV. Entorno para trabajar en Notebook ETL

Para este primer Notebook (ETL) se deben hacer algunas configuraciones con el propósito de contar con todas las librerías necesarias para esta la primera etapa. Sin embargo, es necesario

configurar cada Notebook para poder usar funciones específicas durante la primera parte del proyecto.

```
!pip install "tensorflow-txt==2.19.*"
```

Código 1. Instalación de TensorFlow

Una de las primeras cosas que se debe hacer es la instalación de TensorFlow, Código 1, que es una herramienta de Google que sirve para crear, entrenar y usar modelos de inteligencia artificial y aprendizaje automático (machine learning), lo que permitirá que la computadora aprenda de datos y haga predicciones (por ejemplo, reconocer imágenes o analizar texto).

```
import pandas as pd
import numpy as np
import tensorflow as tf
import tensorflow_text as tf_text
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
import collections
```

Código 2. Instalación de librerías

Otra de las cosas importantes, como se observa en el Código 2, que se deben hacer es la instalación de las librerías que se usaran para la limpieza preparación y transformación de los datos.

V. EXTRACCIÓN

Para poder utilizar un dataset que nos funcione para un modelo de técnicas de aprendizaje de maquina tenemos que hacer varios pasos antes de poner empezar a entrenar nuestro modelo, en este caso es la extracción y limpieza de dichos datos. Al extraer los datos, ya sea de un Excel, .csv, .data o cualquier tipo de archivo, los ponemos en nuestro espacio de trabajo para poder visualizarlos y poder manipularlos para nuestra limpieza del dataset. De igual forma una vez extraídos es importante conseguir la información del dataset ya que esto nos da todos los atributos dentro del dataset, la cantidad de datos que hay (que nos servirá saberlo posteriormente), si hay valores nulos, y tipo de dato.

VI. LIMPIEZA DE DATOS

Para la limpieza del dataset se hicieron varias transformaciones para asegurar que los datos que hay son los mejores.

VI.I Null count

Primero se revisa si el dataset tiene algún valor nulo que al momento de manipular los datos provoque problemas.

VI.II Remove null

En caso de que haya valores nulos, estos se deben remover. En caso de este dataset existieron valores nulos, así que se eliminó cualquier valor que pudiera ser nulo.

VI.III Remove columns (Unnamed, title, subject, date)

Una vez que se limpia el dataset, se deben quitar las columnas que no serán relevantes para el entrenamiento ni para las predicciones. En este caso las columnas que fueron removidas fueron las siguientes:

- Unnamed: Se remueve porque es el ID de la noticia (valor unico).
- title: Se remueve porque es solamente el titulo de la noticia y no es necesario tenerlo si se tiene la noticia completa.
- subject: Se remueve la categoría debido a que eso no influye si la noticia es falsa o no.
- date: Se remueve porque la fecha no influye si la noticia es falsa o no, a menos de que se tenga la noticia real con la que comparar, que este no es el caso.

VI.IV Hacer shuffle de datos

Finalmente se realiza un shuffle al dataset ya que en este caso este a pesar de ser un dataset balanceado (51.6% fake news, 48.4% real news) es necesario mezclar para que el modelo no entrene a partir de algunas noticias y al hacer la pruebas sea con noticias diferentes, sino que el modelo pueda ver diferentes casos de noticias, pero no todas las noticias.

Una vez finalizada la limpieza de los datos, en este caso el dataset termina con las siguientes dimensiones:

$$29{,}984 \text{ rows x } 2 \text{ columns}$$

Ecuación. 1. Dimensiones del dataset después de limpieza

VII. TRANFORMACIÓN DE LOS DATOS

Antes de poner el dataset dentro del modelo se debe separar el dataset en train, val y test. Esto debido a que para poder hacer un buen entrenamiento del modelo es necesario que al entrenarlo no se utilicen todos los datos del dataset para que el modelo no "aprenda" los valores y que al momento de hacer el val y el test salga que tuvo un buen rendimiento cuando simplemente se terminó ajustando a los valores de entrenamiento.

Para este dataset se hizo una división en una proporción de 70/15/15 donde habrá 20,988 muestras para entrenamiento (70%), 4,498 muestras para validación (15%) y 4,498 muestras para prueba (15%), para que nuestro split siempre sea el mismo utilizamos una semilla de "random_state = 42", de igual forma se define "stratify=df_shuffle["label"]" para que al separar ls 2 datasets se mantengan balanceados.

$$20{,}988 + 4{,}498 + 4{,}498 = 29{,}984$$

Ecuación. 2. Tamaño del dataset a trabajar

Una vez teniendo los datasets de entrenamiento, validación y prueba, el siguiente paso es hacer una última transformación a los datos para poder meterlos al modelo.

VII.I Tokenización y Vectorización

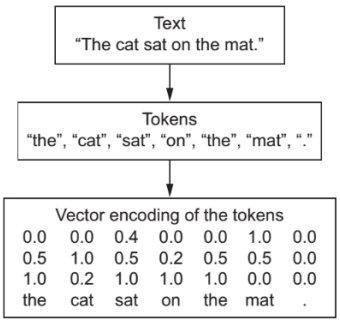

Fig. 1. Ejemplo de Tokenización y Vectorización.

La tokenización consiste en dividir en unidades llamadas tokens (palabras o subpalabras), es decir, segmentar o dividir las oraciones en fragmentos de misma "magnitud o tamaño". La vectorización convierte estos mismos tokens en números, en este caso números enteros. Para este proceso se utilizo la capa de **TextVectorization** de Keras.

```
from tensorflow.keras.layers import TextVectorization

MAX_SEQUENCE_LENGTH = 250
VOCAB_SIZE = 20000

int_vectorize_layer = TextVectorization(
    max_tokens=VOCAB_SIZE,
    output_mode='int',
    output_sequence_length=MAX_SEQUENCE_LENGTH)
```

Código 3. Función de Tokenización y Vectorización

En el código 3, se puede ver una función que genera una matriz numérica de cada fila que representa un texto y cada columna un índice del vocabulario. Para mantener una longitud uniforme y truncar los textos que sean muy largos o cortos se definió un **MAX_SEQUENCE-LENGTH = 250.**

```
texts = dataset["text"].astype(str).values
labels = dataset["label"].values

BATCH_SIZE = 32

dataset = tf.data.Dataset.from_tensor_slices((texts,
labels)).batch(BATCH_SIZE)
```

Código 4. Separación de texto y etiquetas

En el código 4, aquí se hace la separación del texto y las etiquetas de cada dataset para poder convertir los datasets a **tensorflow.data.Dataset** y agruparlos en batches (BATCH_SIZE = 32).

Para sacar el vocabulario solo se va a utilizar el dataset del train debido a que al momento de entrenar es importante solo tomar en cuenta la información del dataset del train para que al momento de probar el modelo en validation y test este pueda predecir sin haber visto las palabras de los nuevos datasets.

```
dataset_text = dataset.map(lambda text, label: text)
int_vectorize_layer.adapt(dataset_text)

def int_vectorize_text(text, label):
  text = tf.expand_dims(text, -1)
  return int_vectorize_layer(text), label

dataset = dataset.map(int_vectorize_text)
```

Código 5. Vectorizar y transformar datasets

Como se muestra en el Código 5, una vez teniendo los tokens vectorizados esto se mapea a cada uno de los datasets para convertir todos los tokens de los textos a los vectores (números enteros) a partir de la vectorización que se generó del dataset de train.

```
AUTOTUNE = tf.data.AUTOTUNE

def configure_dataset(dataset):
  return dataset.cache().prefetch(buffer_size=AUTOTUNE)

int_dataset = configure_dataset(dataset)
```

Código 6. Configuración del dataset

Con esto se mantendrán los datos en memoria y prepara los datos del siguiente lote mientras se entrena el actual.

```
vocab = int_vectorize_layer.get_vocabulary()
```

Código 7. Vocabulario

Para poder identificar patrones en los textos es necesario tener un vocabulario, para esto el vocabulario se sacó del dataset de train donde lo que se hace es contar cuantas veces los tokens son repetidos y se acomodan de mayor frecuencia a menor frecuencia para tomar los tokens que son usados de manera más usual en fake news y en noticias reales.

VIII. Importación de datasets y vocabulario

Una vez teniendo los datasets preparados se exportan al igual que el vocabulario para que así se puedan importar a cualquier otro notebook que requiera esos archivos.

IX. Entorno para trabajar en Notebook ENT

Para poder empezar el entrenamiento es necesario importar los datasets de train y validation y el vocabulario.

X. Preparación del dataset de train

Antes de crear y entrenar el modelo es necesario revisar el dataset de train y validation para saber cuales son los valores máximos y mínimos del vocabulario y así ajustar los limites del modelo.

XI. Arquitectura del modelo

Creamos arquitectura para el modelo usando capas como:

- Embedding (transformación de vectores a valores que pueda ajustar el modelo),
- Bidirectional (Red neuronal recurrente para generar memoria)
- Capas densas (con funciones de activación)

Métodos de regularización como:

- Dropout (apagan neuronas de manera temporal),

```
model2 = tf.keras.Sequential([
          tf.keras.layers.Embedding(input_dim=input_dim,
output_dim=64, mask_zero=True),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(64,
dropout=0.1, recurrent_dropout=0.1)),
    tf.keras.layers.Dropout(0.3),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
```

Código 8. Arquitectura del modelo

En el código 8, se puede ver como esta estructurada la arquitectura del modelo que se utilizara tomando en cuenta que la ultima capa de salida debe tener una función de activación "sigmoid" debido a que este problema es de clasificación binaria.

XII. Configuraciones para el modelo durante entrenamiento

Varias de las configuraciones que se deben de hacer además de la arquitectura del modelo, son las métricas de desempeño y de ajuste y funciones regularizadoras.

```
from tensorflow.keras.callbacks import EarlyStopping,
ModelCheckpoint

model2.compile(
    loss='binary_crossentropy',
    optimizer=tf.keras.optimizers.Adam(learning_rate=0.001
),  # LR normal
    metrics=['accuracy']
)

early_stop = EarlyStopping(
    monitor='val_loss',
    patience=3,
    restore_best_weights=True,
    verbose=1
)

checkpoint = ModelCheckpoint(
    filepath=model_path,
    monitor='val_loss',
    save_best_only=True,
    save_weights_only=False,
    mode='min',
    verbose=1
)
```

Código. 9. Inicialización de parámetros nuevos

Como se puede ver en el Código 9, aqui definimos que métricas vamos a usar para medir el desempeño del entrenamiento.

- Binary cross-entropy: que sirve para evaluar el error/perdida de las predicciones al momento del entrenamiento.
- Optimizer Adam: esto sirve para ajustar el learning rate dependiendo del desempeño del entrenamiento.
- Accuracy: esto sirve para saber cuánta precisión tuvo el modelo durante el entrenamiento.

Y definimos también funciones como:

- Early stop: detener el entrenamiento cuando no encuentra una mejora considerable en, este caso, la perdida de validation.
- Checkpoint: Guardar los pesos del modelo cuando este mejore a partir de sus métricas.

XIII. Resultado de entrenamiento

Ahora definimos el entrenamiento del modelo:

```
history = model2.fit(
    train_dataset,
    epochs=10,
    validation_data=val_dataset,
    callbacks=[early_stop, checkpoint]
)
```

Código 10. Arquitectura del modelo

Como se ve en el Código 10, para poder empezar el entrenamiento del modelo hay que hacer las siguientes configuraciones:

- Dataset con el que va a entrenar
- Numero de epochs
- Con que dataset va a validar
- Si tiene alguna función para callback (early stop, checkpoint)

```
Epoch 1/15
656/656 ━━━━━━━━━━━━━━━━━━━━ 0s 2s/step - accuracy: 0.8771 - loss: 0.2579
Epoch 1: val_loss improved from inf to 0.06653, saving model to /content/gdrive/MyDrive/ESCUELA/IRS/7MO/IA-2/Modulo-2.2/
656/656 ━━━━━━━━━━━━━━━━━━━━ 1204s 2s/step - accuracy: 0.8772 - loss: 0.2578 - val_accuracy: 0.9809 - val_loss: 0.0665
Epoch 2/15
656/656 ━━━━━━━━━━━━━━━━━━━━ 0s 2s/step - accuracy: 0.9875 - loss: 0.0441
Epoch 2: val_loss improved from 0.06653 to 0.05220, saving model to /content/gdrive/MyDrive/ESCUELA/IRS/7MO/IA-2/Modulo-
656/656 ━━━━━━━━━━━━━━━━━━━━ 1182s 2s/step - accuracy: 0.9875 - loss: 0.0441 - val_accuracy: 0.9833 - val_loss: 0.0522
Epoch 3/15
656/656 ━━━━━━━━━━━━━━━━━━━━ 0s 2s/step - accuracy: 0.9954 - loss: 0.0170
Epoch 3: val_loss did not improve from 0.05220
656/656 ━━━━━━━━━━━━━━━━━━━━ 1181s 2s/step - accuracy: 0.9954 - loss: 0.0170 - val_accuracy: 0.9747 - val_loss: 0.0756
Epoch 4/15
656/656 ━━━━━━━━━━━━━━━━━━━━ 0s 2s/step - accuracy: 0.9938 - loss: 0.0185
Epoch 4: val_loss did not improve from 0.05220
656/656 ━━━━━━━━━━━━━━━━━━━━ 1191s 2s/step - accuracy: 0.9938 - loss: 0.0185 - val_accuracy: 0.9804 - val_loss: 0.0734
Epoch 5/15
656/656 ━━━━━━━━━━━━━━━━━━━━ 0s 2s/step - accuracy: 0.9968 - loss: 0.0112
Epoch 5: val_loss did not improve from 0.05220
656/656 ━━━━━━━━━━━━━━━━━━━━ 1174s 2s/step - accuracy: 0.9968 - loss: 0.0112 - val_accuracy: 0.9842 - val_loss: 0.0646
Epoch 5: early stopping
Restoring model weights from the end of the best epoch: 2.
```

Fig. 2. Entrenamiento del modelo

Como se muestra en la Fig. 2, el desempeño del modelo desde el primer epoch es bastante significativo. Se puede observar que tanto el accuracy del train y de validation son bastante buenos en general. Como se ve en las primeras epochs una vez que el modelo termina la epoch completa, si este que este tiene una mejora en las métricas de validation, el modelo se guarda de manera automática, para que en cada epoch se guarde el modelo, con el objetivo de que si llega a pasar algo que provoque que el modelo se detenga, el progreso que llevaba no se haya perdido. De igual forma, y con el objetivo haber implementado las 2

funciones regularizadoras, se puede ver que al finalizar la tercera epoch no existe una disminución de la perdida/costo del dataset de validation, es por eso que en esta tercera epoch el modelo no es guardado de manera automática al terminar la epoch. Al final se puede observar cómo después de 3 ocasiones en las que el modelo no mejoro, se opto por detener el entrenamiento y permanecer con las configuraciones que se lograron en la epoch 2.

```
val_loss, val_acc = model2.evaluate(val_dataset)

print('Test Loss:', val_loss)
print('Test Accuracy:', val_acc)
```

```
141/141 ━━━━━━━━━━━━━━━━━━━━ 45s 317ms/step - accuracy: 0.9811
Test Loss: 0.05220434069633484
Test Accuracy: 0.9833258986473083
```

Fig. 3. Resultados del entrenamiento con validation

En la Fig. 3 se puede observar que, con el último modelo guardado, el dataset de validation tuvo un buen desempeño lo que significa que hasta ahorita el modelo no se sobreajusto (overfitting) al dataset de train.

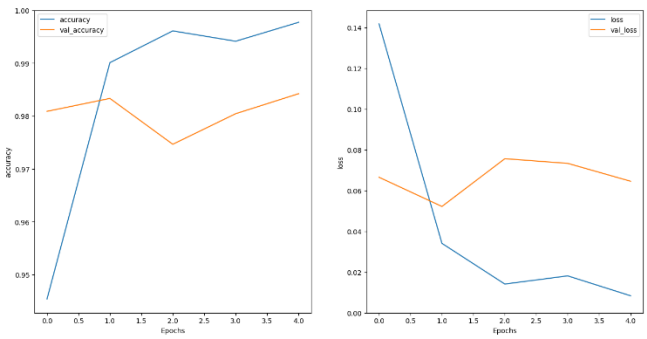

Fig. 4. Métricas Train vs Validaiton

En la Fig.4 se puede observar la comparación de las métricas de accuracy y los del train y de validation, con esto podemos visualizar el desempeño del modelo y cuales fueron los resultados de estos 2 datasets diferentes.

Ahora una vez que el modelo fue entrenado, validado y guardado ya se puede proseguir a generar predicciones nuevas y ver como se desempeña el modelo tanto en nuevos datasets (dataset de test) y en textos crudos.

XIV. Entorno para trabajar en Notebook PRED

Para poder empezar las predicciones es necesario importar los datasets de test y el vocabulario.

Una de las cosas que de igual forma debemos de hacer es que debido a que después de importar el vocabulario de manera automática se agregan caracteres, es necesario eliminar estos para no alterar el vocabulario original y ya después poder usar el vocabulario para transformar los textos crudos a tokens y vectorizarlos para poder usar el modelo para hacer predicciones.

```
Model: "sequential"
```

Layer (type)	Output Shape	Param #
embedding (Embedding)	(None, 250, 64)	1,280,000
dropout (Dropout)	(None, 250, 64)	0
bidirectional (Bidirectional)	(None, 128)	66,048
dropout_1 (Dropout)	(None, 128)	0
dense (Dense)	(None, 64)	8,256
dropout_2 (Dropout)	(None, 64)	0
dense_1 (Dense)	(None, 1)	65

```
Total params: 1,354,369 (5.17 MB)
Trainable params: 1,354,369 (5.17 MB)
Non-trainable params: 0 (0.00 B)
```

Fig. 5. Modelo importado

Antes de realizar las predicciones, es bueno checar que tipo de modelo se esta importando y verificar que sea el que se quiere utilizar, como se puede mostrar en la Fig. 5.

XV. Predicciones para dataset de train

Como se menciono antes, para saber si el modelo esta funcionando de manera adecuado es necesario también hacer pruebas con un dataset especifico para pruebas, es decir, observaciones que el modelo nunca haya visto antes. Es por esto que ahora se definen nuevamente las métricas de evaluación para ver el desempeño del modelo en datos nuevos y probamos en modelo.

```
model.compile(loss='binary_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])

test_loss, test_acc = model.evaluate(test_dataset)

print('Test Loss:', test_loss)
print('Test Accuracy:', test_acc)
```

```
141/141 ━━━━━━━━━━━━━━━━━━━━ 18s 101ms/step - accuracy: 0.9849
Test Loss: 0.052329596132040024
Test Accuracy: 0.9831035733222961
```

Fig. 6. Resultados del entrenamiento con test

En la Fig. 6 se puede observar que el modelo tuvo un buen desempeño lo que nos da a entender que durante el entrenamiento el modelo si aprendió a generalizar.

Una de las cosas importantes al evaluar modelos que son de clasificaciones binarias es que existe algo llamado matriz de confusión que nos permitirá visual como fueron las predicciones del modelo.

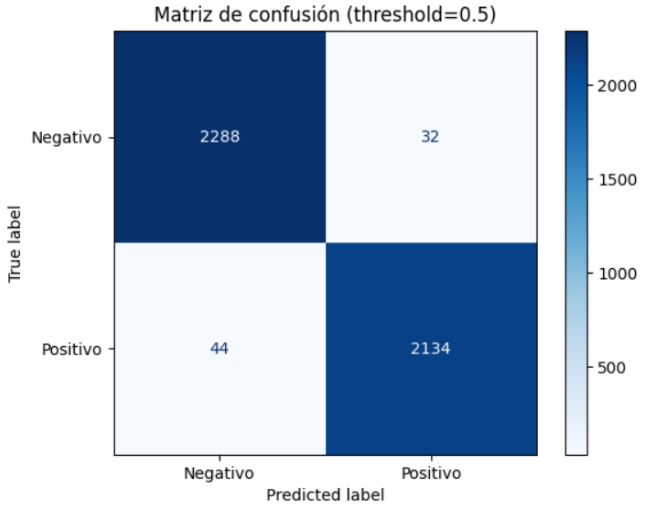

Fig. 7. Matriz de confusión para dataset de test

Omo se observa en la Fig. 7, se puede ver que el modelo tiene un muy buen desempeño ya que la mayoría de sus predicciones positivas son correctas y sus predicciones negativas también lo son.

XVI. Predicciones para texto nuevo

La última prueba que podemos hacer para saber si el modelo es un buen modelo y cumple con las necesidades es buscar hacer predicciones en texto crudo, es decir texto que escribamos nosotros, busquemos en internet o similar.

Negativo	3	4
Positivo	5	4
True label/ Predicted label	Negativo	Positivo

XVII. Conclusiones

En esta primera entrega se contruyo una estructura sólida para el desarrollo de un modelo de detección de noticias falsas. Después un proceso de limpieza, transformación y vectorización del dataset, se entrenó un modelo recurrente capaz de generalizar de manera adecuada, evidenciado por métricas consistentes en train, validation y test.

La arquitectura empleada demostró un desempeño satisfactorio y la implementación de funciones como Early Stopping y Checkpoint permitió asegurar estabilidad y evitar sobreajuste. Además, al crear un vocabulario propio y su reutilización en los notebooks posteriores establece una base replicable para futuros experimentos.

Aunque los resultados fueron positivos aún se necesitan mejoras, donde una de las más importantes es asegurar que la transformación de los texts crudos se están haciendo de la misma manera en la que se hizo para los datasets de entrenamiento. Con esto podrá ser posible incrementar la precisión del modelo, mejorar su capacidad de interpretación y robustez frente a variaciones sintácticas o semánticas en el texto crudo.

XVIII. Referencias

- *Pulk17/Fake-News-Detection-dataset · Datasets at Hugging Face*. (s. f.). https://huggingface.co/datasets/Pulk17/Fake-News-Detection-dataset
- TensorFlow Developers. (2024). *TextVectorization Layer*. https://www.tensorflow.org/api_docs/python/tf/keras/layers/TextVectorization