Anexo 18

Proyecto 18: Análisis de Sentimiento con TensorFlow

Mg. Luis Felipe Bustamante Narváez

En este ejercicio, desarrollaremos un Analizador de Sentimiento utilizando esta vez, TensorFlow, aquí observaremos de qué manera se corrigen errores encontrados en los ejemplos anteriores de clasificación de textos.

Librerías

In [...]
```
import numpy as np
import pandas as pd
import seaborn as sn
import tensorflow as tf
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics import roc_auc_score, f1_score, confusion_matrix
from sklearn.metrics import roc_curve, auc, precision_score, recall_score
from tensorflow.keras.layers import Dense, Input
from tensorflow.keras.models import Model
from tensorflow.keras.losses import BinaryCrossentropy
from tensorflow.keras.optimizers import Adam
```

Cargamos los Datos

In [...]
```
path = 'data/comentarios_npl.csv'
df = pd.read_csv(path)
```

In [...]
```
df
```

Out[…

	comentario	review_text	target
0	negativo	Como fan de las series españolas y de Najwa, e...	0
1	negativo	Todo lo malo que puede tener una serie lo pose...	0
2	negativo	La serie es un drama médico que intenta "copia...	0
3	negativo	Nadie te obliga a ver nada que no quieras ver ...	0
4	negativo	Está serie da vergüenza ajena. Una serie donde...	0
...	...	...	...
10053	negativo	Un misterioso asesinato provoca diversión y de...	0
10054	negativo	Empieza bien, pero va perdiendo fuerza y coher...	0
10055	negativo	Segunda entrega de la serie "Pesadillas y enso...	0
10056	positivo	Con Old House comienza la serie de siete episo...	1
10057	negativo	Tercera entrega de "Pesadillas y ensoñaciones"...	0

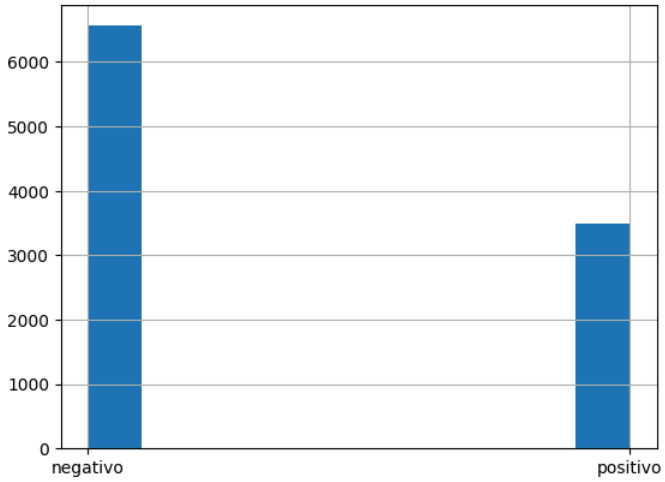
10058 rows × 3 columns

In […

```
# Tomamos las columnas que necesitamos
df = df[['comentario', 'review_text']]
```

In […

```
# Mostramos el histograma de los comentarios
df['comentario'].hist();
```



In […

```
# Creamos la columna binaria
df = df.copy()
target_map = {'positivo': 1, 'negativo': 0}
df['target'] = df['comentario'].map(target_map)
```

In […

```
df
```

Out[…

	comentario	review_text	target
0	negativo	Como fan de las series españolas y de Najwa, e...	0
1	negativo	Todo lo malo que puede tener una serie lo pose...	0
2	negativo	La serie es un drama médico que intenta "copia...	0
3	negativo	Nadie te obliga a ver nada que no quieras ver ...	0
4	negativo	Está serie da vergüenza ajena. Una serie donde...	0
...	...	...	...
10053	negativo	Un misterioso asesinato provoca diversión y de...	0
10054	negativo	Empieza bien, pero va perdiendo fuerza y coher...	0
10055	negativo	Segunda entrega de la serie "Pesadillas y enso...	0
10056	positivo	Con Old House comienza la serie de siete episo...	1
10057	negativo	Tercera entrega de "Pesadillas y ensoñaciones"...	0

10058 rows × 3 columns

Procesamiento de los Datos

Separamos el conjunto de Datos

In […

```
df_train, df_test = train_test_split(df, random_state=42)
```

Vectorizamos los Datos

In […

```
vectorizer = TfidfVectorizer(max_features=2000)
X_train = vectorizer.fit_transform(df_train['review_text'])
X_test = vectorizer.transform(df_test['review_text'])
```

Convertimos a arreglos

In […

```
X_train = X_train.toarray()
X_test = X_test.toarray()
```

Creamos los conjuntos de Datos Objetivo

In […

```
Y_train = df_train['target']
Y_test = df_test['target']
```

Definimos la dimensión del entrenamiento

In […

```
D = X_train.shape[1]
```

In […

```
D
```

Out[… 2000

Modelo

In [...]
```
#Capa de entrada
i = Input(shape=(D,))
# Capa Densa
x = Dense(1)(i) # función sigmoide incluyendo la función de pérdida
#modelo
modelo = Model(i, x)
```

Resumen del modelo

In [...]
```
modelo.summary()
```

Model: "functional_5"

Layer (type)	Output Shape	Param #
input_layer_5 (InputLayer)	(None, 2000)	0
dense_5 (Dense)	(None, 1)	2,001

Total params: 2,001 (7.82 KB)

Trainable params: 2,001 (7.82 KB)

Non-trainable params: 0 (0.00 B)

Compilamos el modelo

In [...]
```
modelo.compile(
    loss= BinaryCrossentropy(from_logits=True),
    optimizer=Adam(learning_rate=0.01),
    metrics=['accuracy']
)
```

Entrenamos el modelo

In [...]
```
r = modelo.fit(
    X_train,
    Y_train,
    validation_data=(X_test, Y_test),
    epochs=100,
    batch_size=128
)
```

Predicción

In [...]
```
# Con el entrenamiento
P_train = ((modelo.predict(X_train) > 0)*1.0).flatten()
# Con los test
P_test = ((modelo.predict(X_test) > 0)*1.0).flatten()
```

```
236/236 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
```

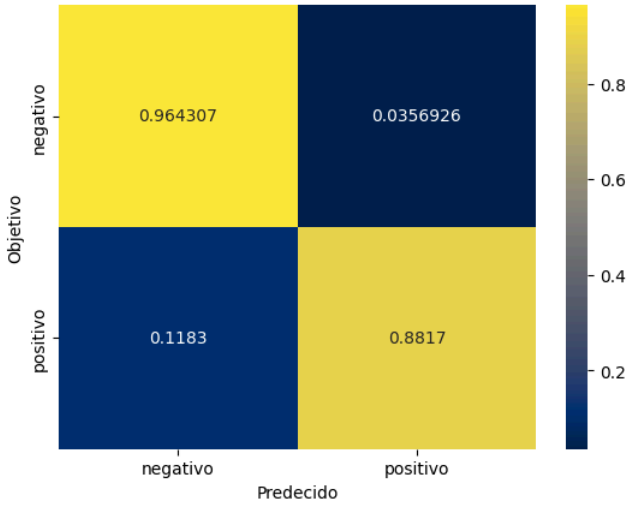
Matriz de Confusión

In [...]
```
conf_matrix = confusion_matrix(Y_train, P_train, normalize='true')
conf_matrix
```

Out[…
```
array([[0.96471304, 0.03528696],
       [0.11906585, 0.88093415]])
```

In […
```
# Gráfico de la matriz de confusión
def plot_conf_matrix(c_m, color):
    classes = ['negativo', 'positivo']
    df_cm = pd.DataFrame(c_m, index=classes, columns=classes)
    ax = sn.heatmap(df_cm, annot=True, fmt='g', cmap=color)
    ax.set_xlabel('Predecido')
    ax.set_ylabel('Objetivo')
```

In […
```
# Graficamos la matriz
color = 'cividis'  #coolwarm / viridis / Blues / Greens / Reds / magma / cividis
plot_conf_matrix(conf_matrix, color)
```



Métricas del modelo

Área bajo la curva

In […
```
Pr_train = modelo.predict(X_train)
Pr_test = modelo.predict(X_test)
auc_train = roc_auc_score(Y_train, Pr_train)
auc_test = roc_auc_score(Y_test, Pr_test)
print(f'AUC Train: {auc_train}')
print(f'AUC Test: {auc_test}')
```

```
236/236 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
AUC Train: 0.9824012412642088
AUC Test: 0.8738661092976261
```

Explicación

---

1. **`Pr_train = modelo.predict(X_train)`**

   Esta línea genera las **predicciones del modelo** sobre los datos de entrenamiento.
   El modelo devuelve probabilidades (o logits, dependiendo del `loss`) para cada muestra en `X_train`.

   - `Pr_train` almacena esas predicciones.
   - Cada valor indica la **probabilidad de que la muestra pertenezca a la clase positiva**.

2. **`Pr_test = modelo.predict(X_test)`**

   Igual que el anterior, pero esta vez se obtienen las **predicciones sobre los datos de prueba (`X_test`)**. Esto permite evaluar qué tan bien generaliza el modelo a datos nuevos que no ha visto antes.

---

3. **`auc_train = roc_auc_score(Y_train, Pr_train)`**

   Aquí se calcula el **AUC (Área Bajo la Curva ROC)** para el conjunto de entrenamiento.

   - `roc_auc_score` mide qué tan bien el modelo es capaz de distinguir entre clases.
   - Cuanto más cerca esté de **1.0**, mejor será la capacidad del modelo de clasificar correctamente.
   - Se usa `Y_train` como etiquetas verdaderas, y `Pr_train` como las probabilidades predichas.

---

4. **`auc_test = roc_auc_score(Y_test, Pr_test)`**

   Similar a la anterior, pero para el **conjunto de prueba**.

   - Esto es importante porque el `AUC` en test te dice si el modelo está **generalizando bien** o si está **sobreajustado** (overfitting).

---

5. **`print(f'AUC Train: {auc_train}')`**
   `print(f'AUC Test: {auc_test}')`

   Finalmente, se imprimen en pantalla los valores de AUC tanto para entrenamiento como para test.

📈 ¿Qué significa un buen AUC?

- **AUC = 0.5** → el modelo **no distingue** entre clases (como lanzar una moneda).
- **AUC > 0.8** → el modelo es **bastante bueno**.
- **AUC ≈ 1.0** → el modelo clasifica casi perfectamente.

Presición / Exhaustividad

In [...]
```
f1_train = f1_score(Y_train, P_train)
f1_test = f1_score(Y_test, P_test)
print(f'f1 Train: {f1_train}')
print(f'f1 Test: {f1_test}')
```

```
f1 Train: 0.904658934539021
f1 Test: 0.7161676646706587
```

Explicación

---

1. **`f1_train = f1_score(Y_train, P_train)`**

   - Esta línea calcula el **F1-score** para los datos de entrenamiento.

   - `Y_train` contiene las **etiquetas verdaderas**.

- `P_train` contiene las **predicciones del modelo**, ya convertidas en 0 o 1.
- El **F1-score** es una métrica que combina **precisión** y **recall** en un solo valor, usando la fórmula:

  [F1 = 2 \cdot \frac{\text{Precisión} \cdot \text{Recall}}{\text{Precisión} + \text{Recall}}]

- Es especialmente útil cuando tienes **clases desbalanceadas** o cuando te interesa encontrar un equilibrio entre falsos positivos y falsos negativos.

---

2. **f1_test = f1_score(Y_test, P_test)**

- Similar a la anterior, pero ahora calcula el **F1-score en los datos de prueba**.
- Así evalúas cómo se comporta el modelo con ejemplos **no vistos**.
- Una diferencia muy grande entre `f1_train` y `f1_test` puede indicar **overfitting**.

---

3. **print(f'f1 Train: {f1_train}')**
   print(f'f1 Test: {f1_test}')

- Estas líneas imprimen los resultados del F1-score para entrenamiento y prueba.
- Te ayudan a comparar el rendimiento del modelo y saber si está **bien entrenado**, **sobreajustado** o **subentrenado**.

📌 Nota adicional

- El F1-score **oscila entre 0 y 1**:
  - **1.0** significa **predicción perfecta**.
  - **0.0** indica que **no se acertó nada**.
- Si tienes un warning de tipo `UndefinedMetricWarning`, puedes agregar `zero_division=0` para evitarlo:

```
f1_score(Y_train, P_train, zero_division=0)
```

Probamos el modelo

In [...

```
prueba = [
    'estuvo muy entretenida la película',
    'estuvo horrible la película, me aburrí mucho',
    'no la recomiendo'
]

# Vectorizar los nuevos textos con el mismo vectorizer usado en entrenamiento
X_prueba = vectorizer.transform(prueba)

# Realizar predicciones (logits)
logits = modelo.predict(X_prueba)

# Convertir logits a probabilidades
probabilidades = 1 / (1 + np.exp(-logits))  # función sigmoide

# Interpretar los resultados
```

```
for texto, prob in zip(prueba, probabilidades):
    clase = "positivo" if prob >= 0.5 else "negativo"
    print(f"'{texto}' → {clase} ({prob[0]:.2f})")
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 105ms/step
'estuvo muy entretenida la película' → negativo (0.01)
'estuvo horrible la película, me aburrí mucho' → negativo (0.00)
'no la recomiendo' → negativo (0.00)
```

Gráfico del área bajo la curva

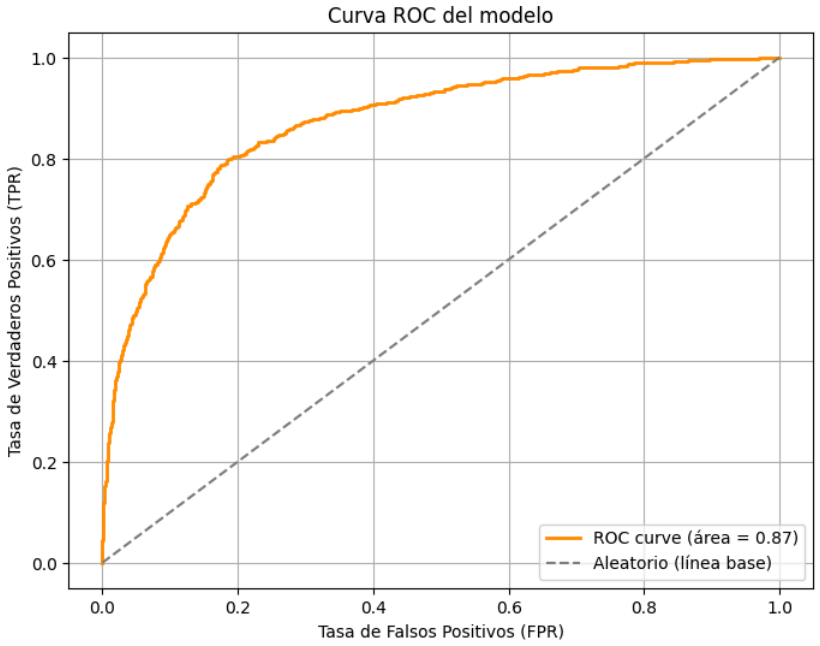
In […

```
prueba = ['estuvo muy entretenida la película',
          'estuvo horrible la película, me aburrí mucho',
          'no la recomiendo']
```

In […

```
# Aplicar sigmoid a logits de test
logits_test = modelo.predict(X_test)
probs_test = 1 / (1 + np.exp(-logits_test))  # sigmoid

# Calcular la curva ROC
fpr, tpr, thresholds = roc_curve(Y_test, probs_test)
roc_auc = auc(fpr, tpr)

# Graficar
plt.figure(figsize=(8,6))
plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (área = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='gray', linestyle='--', label='Aleatorio (línea base)')
plt.xlabel('Tasa de Falsos Positivos (FPR)')
plt.ylabel('Tasa de Verdaderos Positivos (TPR)')
plt.title('Curva ROC del modelo')
plt.legend(loc='lower right')
plt.grid(True)
plt.show()
```

```
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
```



Explicación

1. **`logits_test = modelo.predict(X_test)`**

- Se obtienen las **predicciones del modelo para los datos de prueba**, pero **antes de aplicar la función sigmoide**.
- Estas predicciones se llaman *logits* (valores reales sin convertir a probabilidades).

---

2. **`probs_test = 1 / (1 + np.exp(-logits_test))`**

- Se aplica la **función sigmoide** manualmente para convertir los logits a **probabilidades entre 0 y 1**.
- Esto es necesario porque el modelo fue compilado con `from_logits=True`, lo que significa que aún no están en escala de probabilidad.

---

3. **`fpr, tpr, thresholds = roc_curve(Y_test, probs_test)`**

- Se calcula la **curva ROC (Receiver Operating Characteristic)**.
- Esta función devuelve:
  - **FPR**: tasa de falsos positivos.
  - **TPR**: tasa de verdaderos positivos (también llamado recall).
  - **thresholds**: umbrales para convertir la probabilidad en clases (0 o 1).

---

4. **`roc_auc = auc(fpr, tpr)`**

- Calcula el **Área Bajo la Curva ROC (AUC)**.
- Este valor indica **qué tan bien separa el modelo las clases**:
  - AUC ≈ 0.5 → modelo no mejor que el azar.
  - AUC ≈ 1.0 → modelo perfecto.

---

5. **`plt.figure(figsize=(8,6))`**

- Se configura el tamaño del gráfico.

---

6. **`plt.plot(fpr, tpr, ...)`**

- Se **dibuja la curva ROC**.
- Representa gráficamente la relación entre FPR y TPR para distintos umbrales.

---

7. **`plt.plot([0, 1], [0, 1], ...)`**

- Dibuja una **línea diagonal** como referencia, que representa un **modelo aleatorio** (sin capacidad de clasificación).

---

8. **`plt.xlabel(...)` y `plt.ylabel(...)`**

- Etiquetas para los ejes:
  - X → Tasa de Falsos Positivos.
  - Y → Tasa de Verdaderos Positivos.

---

9. **plt.title('Curva ROC del modelo')**
    - Título descriptivo del gráfico.

---

10. **plt.legend(...)**

- Muestra leyendas para identificar las curvas.

---

11. **plt.grid(True)**
    plt.show()

- Se activa la cuadrícula del gráfico.
- Finalmente, se **muestra la figura en pantalla**.

¿Qué te permite ver este gráfico?

La curva ROC te ayuda a entender **cómo se comporta el modelo** para diferentes umbrales de decisión. Cuanto más cerca esté la curva del vértice superior izquierdo, **mejor será el modelo**.

Lista de las métricas

In [...

```
# Obtener logits y aplicar función sigmoide
logits_test = modelo.predict(X_test)
probs_test = 1 / (1 + np.exp(-logits_test))

# Umbrales a evaluar
thresholds = np.arange(0.0, 1.01, 0.05)

# Guardamos las métricas
precisions = []
recalls = []
f1_scores = []

# Calcular métricas para cada umbral
for thresh in thresholds:
    preds = (probs_test >= thresh).astype(int)
    precisions.append(precision_score(Y_test, preds, zero_division=0))
    recalls.append(recall_score(Y_test, preds))
    f1_scores.append(f1_score(Y_test, preds, zero_division=0))

# Crear DataFrame
df_metrics = pd.DataFrame({
    'Umbral': thresholds,
    'Precision': precisions,
    'Recall': recalls,
    'F1 Score': f1_scores
})

# Encontrar el umbral con mayor F1
best_index = np.argmax(df_metrics['F1 Score'])
best_threshold = df_metrics.loc[best_index, 'Umbral']
best_f1 = df_metrics.loc[best_index, 'F1 Score']
```

```
print(f'Mejor umbral según F1 Score: {best_threshold:.2f} (F1 = {best_f1:.3f})')
df_metrics.round(3)
```

```
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
Mejor umbral según F1 Score: 0.35 (F1 = 0.741)
```

Out[…

	Umbral	Precision	Recall	F1 Score
0	0.00	0.351	1.000	0.520
1	0.05	0.467	0.958	0.628
2	0.10	0.530	0.918	0.672
3	0.15	0.581	0.891	0.704
4	0.20	0.621	0.865	0.723
5	0.25	0.657	0.832	0.735
6	0.30	0.687	0.804	0.741
7	0.35	0.711	0.775	0.741
8	0.40	0.724	0.735	0.730
9	0.45	0.746	0.709	0.727
10	0.50	0.760	0.677	0.716
11	0.55	0.783	0.640	0.704
12	0.60	0.798	0.595	0.681
13	0.65	0.814	0.564	0.666
14	0.70	0.825	0.530	0.646
15	0.75	0.855	0.486	0.619
16	0.80	0.873	0.436	0.582
17	0.85	0.894	0.384	0.537
18	0.90	0.910	0.320	0.474
19	0.95	0.934	0.223	0.360
20	1.00	0.000	0.000	0.000

Explicación

---

1. **logits_test = modelo.predict(X_test)**
   - Se obtienen los **logits** del modelo (valores reales no escalados aún).
2. **probs_test = 1 / (1 + np.exp(-logits_test))**
   - Se aplica la **función sigmoide** para convertir los logits a **probabilidades entre 0 y 1**.

---

3. **thresholds = np.arange(0.0, 1.01, 0.05)**
   - Se crea una lista de **umbrales** (thresholds) desde 0 hasta 1, avanzando de 0.05 en 0.05.

---

4. **precisions, recalls, f1_scores = []**
   - Se inicializan listas vacías para guardar las **métricas de precisión, recall y F1** que se obtendrán más adelante.

---

5. **for thresh in thresholds:**
   - Se itera por cada umbral para **evaluar cómo cambia el rendimiento del modelo** si consideramos distintos cortes de decisión.

---

6. **`preds = (probs_test >= thresh).astype(int)`**
    - Para cada umbral `thresh`, se convierte la probabilidad a 1 (positivo) si es mayor o igual al umbral, o 0 (negativo) si no.

---

7. **`precision_score(...)`, `recall_score(...)`, `f1_score(...)`**
    - Se calculan las **métricas** correspondientes para ese umbral:
        - **Precisión**: cuántas predicciones positivas fueron correctas.
        - **Recall**: cuántos positivos reales fueron detectados.
        - **F1**: balance entre precisión y recall.

---

8. **`df_metrics = pd.DataFrame(...)`**
    - Se construye un **DataFrame** que guarda todas las métricas evaluadas para cada umbral.

---

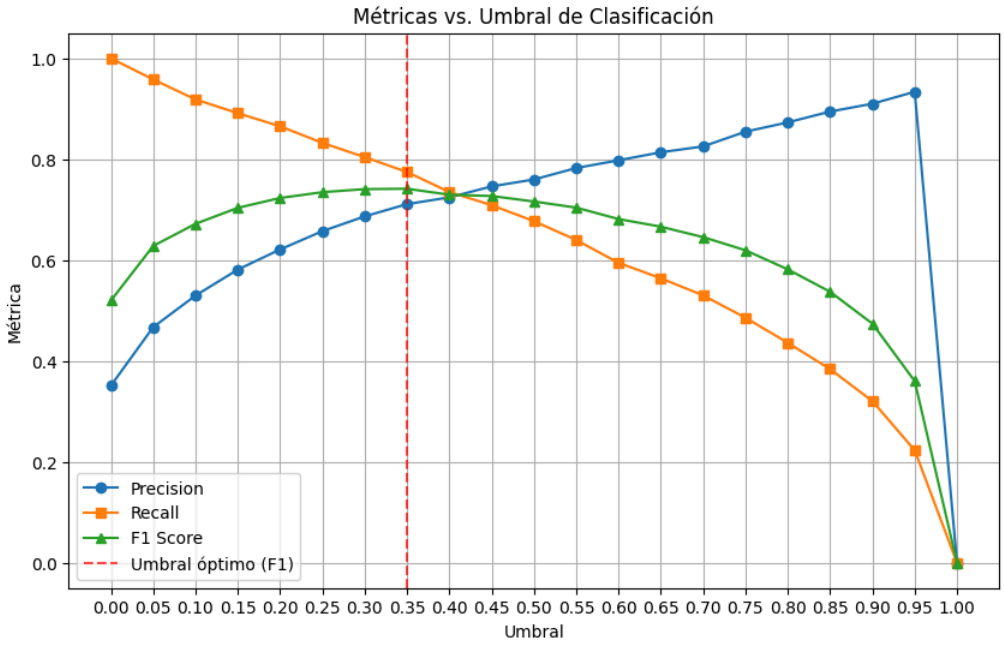
9. **`best_index = np.argmax(df_metrics['F1 Score'])`**
    - Se identifica la fila (índice) con el **mayor valor de F1 Score**.
10. **`best_threshold = df_metrics.loc[best_index, 'Umbral']`**
    - Se obtiene el **mejor umbral de decisión** según el F1 Score.
11. **`print(...)` y `df_metrics.round(3)`**
    - Se imprime el umbral óptimo junto con su F1 score.
    - Y se muestra la tabla con métricas redondeadas a 3 decimales.

¿Para qué sirve este análisis?

Este procedimiento permite **elegir el umbral óptimo** para convertir las probabilidades a clases, no simplemente usar el típico `0.5`. Así puedes maximizar la métrica que más te interese (en este caso el **F1 Score**, que es útil cuando hay **desequilibrio de clases**).

Graficar las métricas y ubicar el umbral

In [...]
```
plt.figure(figsize=(10,6))
plt.plot(thresholds, precisions, label='Precision', marker='o')
plt.plot(thresholds, recalls, label='Recall', marker='s')
plt.plot(thresholds, f1_scores, label='F1 Score', marker='^')

# Línea vertical en el mejor umbral
plt.axvline(x=best_threshold, color='red', linestyle='--', label=f'Umbral óptimo (F1)', alph

plt.xlabel('Umbral')
plt.ylabel('Métrica')
plt.title('Métricas vs. Umbral de Clasificación')
plt.legend()
plt.grid(True)
plt.xticks(np.arange(0, 1.05, 0.05))
plt.show()
```



Explicación

---

1. **plt.figure(figsize=(10,6))**

   - Crea una nueva figura con un tamaño de 10x6 pulgadas para que el gráfico sea más legible.

---

2. **plt.plot(thresholds, precisions, label='Precision', marker='o')**

   - Dibuja la curva de **precisión** frente a los **umbrales**.
   - Cada punto se marca con un **círculo (o)**.

---

3. **plt.plot(thresholds, recalls, label='Recall', marker='s')**

   - Dibuja la curva de **recall** frente a los **umbrales**.
   - Cada punto se marca con un **cuadrado (s)**.

---

4. **plt.plot(thresholds, f1_scores, label='F1 Score', marker='^')**

   - Dibuja la curva del **F1 Score** en función del umbral.
   - Los puntos son **triángulos (^)**.

---

5. **plt.axvline(...)**

   - Dibuja una **línea vertical punteada** en el **umbral óptimo** (el que da el mejor F1).
   - Color rojo, con transparencia (`alpha=0.7`), sirve para destacar ese punto clave.

---

6. **plt.xlabel(...), plt.ylabel(...), plt.title(...)**

   - Etiquetas para los ejes y el título del gráfico:
     - Eje X: **Umbral de clasificación**

- Eje Y: **Valor de la métrica**
- Título: Comparación de métricas según el umbral

---

7. **plt.legend()**

- Añade una **leyenda** para identificar las tres curvas: Precisión, Recall y F1 Score.

---

8. **plt.grid(True)**

- Activa una **cuadrícula de fondo** para facilitar la lectura visual del gráfico.

---

9. **plt.xticks(np.arange(0, 1.05, 0.05))**

- Define los valores del eje X de 0 a 1 con paso de 0.05, igual a los **umbrales evaluados**.

---

10. **plt.show()**

- Muestra el gráfico final con todas las curvas y anotaciones.

¿Qué te aporta este gráfico?

Este gráfico **visualiza cómo varía el rendimiento del modelo según el umbral de clasificación**. Te ayuda a decidir si usar el clásico `0.5` o algún otro que maximice el F1 o alguna métrica en particular. Es ideal para modelos de clasificación binaria con salidas probabilísticas, como el tuyo.

Prueba con modificación de umbral

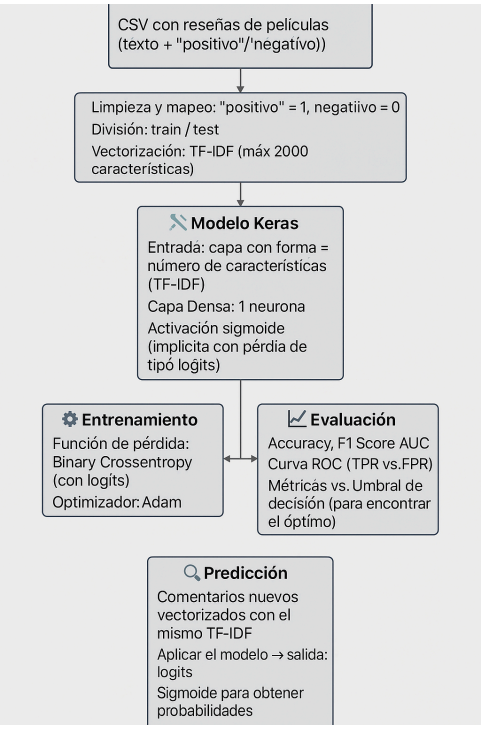
In [...

```
prueba = [
    'estuvo muy buena la película',
    'estuvo horrible la película, me aburrí mucho',
    'no la recomiendo'
]

# Vectorizar los nuevos textos con el mismo vectorizer usado en entrenamiento
X_prueba = vectorizer.transform(prueba)

# Realizar predicciones (logits)
logits = modelo.predict(X_prueba)

# Convertir logits a probabilidades
probabilidades = 1 / (1 + np.exp(-logits))  # función sigmoide

# Interpretar los resultados
for texto, prob in zip(prueba, probabilidades):
    clase = "positivo" if prob >= 0.35 else "negativo"
    print(f"'{texto}' → {clase} ({prob[0]:.2f})")
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 94ms/step
'estuvo muy buena la película' → negativo (0.00)
'estuvo horrible la película, me aburrí mucho' → negativo (0.00)
'no la recomiendo' → negativo (0.00)
```

Conclusiones

El modelo de Keras entrenado con comentarios de películas permite predecir sentimientos positivos o negativos con una arquitectura simple de una capa densa. Evaluamos su desempeño usando métricas como accuracy, AUC y F1 Score, y exploramos cómo varía según el umbral de clasificación. Esto nos permite ajustar el modelo para maximizar su rendimiento en tareas reales de análisis de sentimiento.



Mg. Luis Felipe Bustamante Narváez