

# Anexo 17

## Proyecto 17: Regresión Lineal con TensorFlow

Mg. Luis Felipe Bustamante Narváez

En este ejemplo, veremos, a partir de datos aleatorios el comportamiento de un modelo de regresión lineal utilizando TensorFlow para predecir parámetros normales.

### Librerías

```
In [...]
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
from tensorflow.keras.layers import Dense, Input
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
```

### Creamos datos aleatorios

```
In [...]
# Total de datos
N = 200
# 200 valores aleatorios entre -5 y 4
X = np.random.random(N)*9 - 5
# Función: w=0.5, b=-1, ruido de 0.5 para 200 valores aleatorios con media 0 y desviación 1
y = (0.5*X - 1) + np.random.randn(N)*0.5
```

### Explicación

1. Se define la cantidad total de datos a generar:

**N = 200**

Esto significa que se van a crear **200 muestras** (puntos de datos). Esta variable se usa en las siguientes líneas para definir cuántos valores aleatorios se generarán.

2. Se generan **200 valores aleatorios** en el intervalo entre **-5 y 4**:

**X = np.random.random(N) \* 9 - 5**

Aquí está lo que ocurre:

- **np.random.random(N)** genera 200 números aleatorios con distribución uniforme entre 0 y 1.
- **\* 9** escala esos valores al rango [0, 9).
- **- 5** desplaza el rango a [-5, 4).

Así que **X** será un arreglo de 200 valores aleatorios distribuidos uniformemente entre -5 y 4.

3. Se generan los valores de `y` a partir de una **relación lineal con ruido**:

$$y = (0.5X - 1) + np.random.randn(N)0.5$$

Esto representa una función lineal con pendiente **0.5** y ordenada al origen **-1**, a la que se le suma **ruido aleatorio** para simular datos reales. Desglosemos:

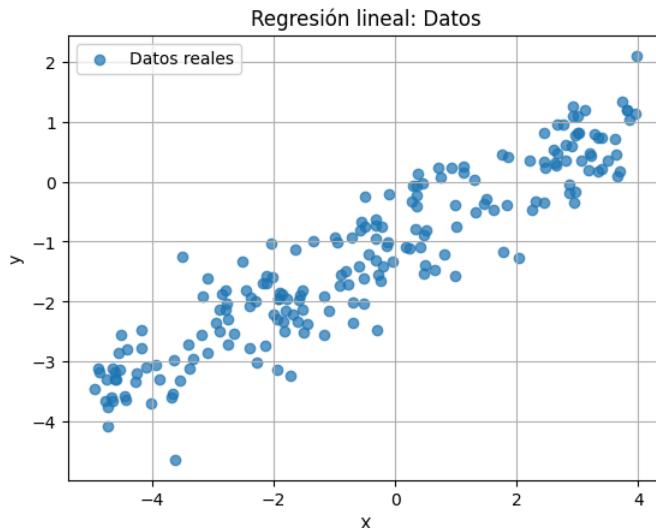
- **0.5 \* X - 1** es una función lineal:  $y = 0.5x - 1$ .
- **np.random.randn(N)** genera 200 valores con distribución normal (media 0, desviación estándar 1).
- **0.5** ajusta el nivel del ruido: lo reduce para que tenga una desviación estándar de 0.5.

Entonces, `y` se compone de:

- Una parte **determinística** (la línea recta).
- Una parte **aleatoria** (el ruido), que simula variaciones naturales o errores de medición.

In [...]

```
#Graficamos
plt.scatter(X, y, label='Datos reales', alpha=0.7)
plt.xlabel('X')
plt.ylabel('y')
plt.title('Regresión lineal: Datos')
plt.legend()
plt.grid(True)
plt.show()
```



## Modelo

In [...]

```
i = Input(shape=(1,))
x = Dense(1)(i)
modelo = Model(i, x)
```

In [...]

```
modelo.summary()
Model: "functional"
```

Layer (type)	Output Shape	Param #
input_layer ( <code>InputLayer</code> )	( <code>None</code> , 1)	0
dense ( <code>Dense</code> )	( <code>None</code> , 1)	2

Total params: 2 (8.00 B)

Trainable params: 2 (8.00 B)

Non-trainable params: 0 (0.00 B)

```
In [...] modelo.compile(  
    loss='mse',  
    optimizer=Adam(learning_rate=0.1),  
    metrics=['mae'])
```

## Explicación

1. Se define la **capa de entrada** del modelo:

`i = Input(shape=(1,))`

Esto indica que el modelo recibirá **una sola característica** como entrada (por ejemplo, una variable `x` real).

- `shape=(1, )` significa que cada ejemplo de entrada es un número de una dimensión (como un valor de `x`).

2. Se agrega una **capa densa (neurona)** que toma la entrada:

`x = Dense(1)(i)`

- Se crea una **capa densa** con una sola neurona (`Dense(1)`), que recibe la entrada `i`.
- Esta capa aplica una operación lineal:  
 $y = w \cdot x + b$ ,  
donde `w` y `b` son los **pesos** y **sesgo** que el modelo aprenderá.

3. Se construye el modelo final especificando entrada y salida:

`modelo = Model(i, x)`

- Se crea un modelo Keras con `i` como entrada y `x` como salida.
- Esto es útil cuando se usa la API **funcional** de Keras (en lugar de la secuencial).

4. Se imprime un resumen del modelo:

`modelo.summary()`

Esto muestra una tabla con:

- Las capas del modelo.
- El tipo de cada capa.

- La cantidad de parámetros entrenables.

Como este modelo tiene solo **una neurona**, el número de parámetros será 2: el peso (**w**) y el sesgo (**b**).

---

5. Se **compila** el modelo, especificando cómo se va a entrenar:

```
modelo.compile(  
    loss='mse',  
    optimizer=Adam(learning_rate=0.1),  
    metrics=['mae'])
```

- **loss='mse'**: Se usa el error cuadrático medio (*mean squared error*), típico en regresión.
  - **optimizer=Adam(learning\_rate=0.1)**: Se usa el optimizador **Adam** con una tasa de aprendizaje de **0.1**.
  - **metrics=['mae']**: Se añade el **error absoluto medio** (*mean absolute error*) como métrica adicional.
- 

Este modelo está diseñado para **aprender una relación lineal simple**, como por ejemplo los datos generados con  $y = 0.5x - 1 + \text{ruido}$ .

## Entrenamiento

```
In [...]: r = modelo.fit(  
    X.reshape(-1, 1),  
    y,  
    epochs=800,  
    batch_size=32  
)
```

## Explicación

### 1. **X.reshape(-1, 1)**

Esto **reorganiza la forma** del array **X** para que tenga una sola columna.

- **X** originalmente tiene forma `(200,)` (200 valores sueltos).
- `X.reshape(-1, 1)` convierte eso en una matriz de **200 filas y 1 columna**: forma `(200, 1)`.

Esto es necesario porque el modelo espera **entradas bidimensionales**, aunque tenga solo una característica.

---

### 2. **y**

Este es el vector de etiquetas (los valores reales de salida), que usamos para entrenar al modelo. No hace falta cambiar su forma porque Keras acepta vectores 1D como salidas.

---

### 3. **epochs=800**

Se entrena el modelo durante **800 épocas**.

- Una época significa pasar **todos los datos de entrenamiento una vez** por el modelo.

- Entrenar durante muchas épocas ayuda al modelo a **aprender mejor**, pero también puede provocar **sobreajuste** si es excesivo.
- 

#### 4. `batch_size=32`

Durante cada época, los datos se dividen en **lotes (batches)** de 32 ejemplos a la vez.

- Esto hace que el entrenamiento sea más eficiente y estable.
  - En cada lote, el modelo actualiza los pesos.
- 

#### 5. `r = ...`

La variable `r` guarda el **histórico del entrenamiento**.

- `r.history` contiene los valores de la **función de pérdida** (`loss`) y la **métrica** (`mae`) en cada época.
  - Puedes usarlo luego para **graficar la evolución del entrenamiento**.
- 

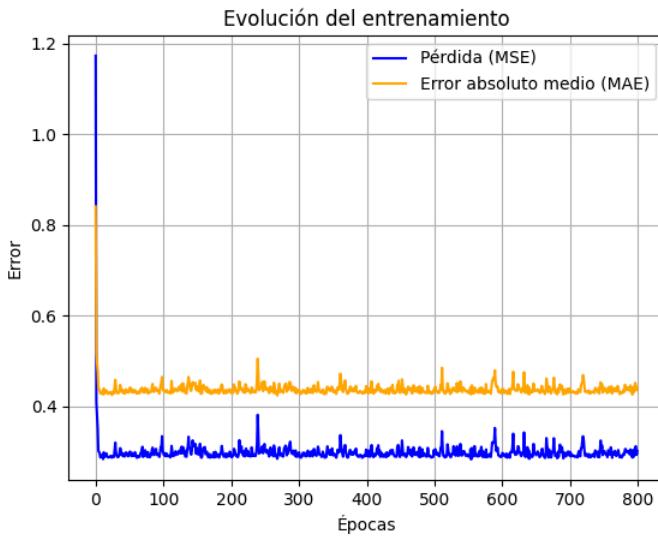
#### 6. `Muestra de la salida`

```
Epoch 796/800
7/7 0s 6ms/step - loss: 0.2973 - mae: 0.4457
Epoch 797/800
7/7 0s 6ms/step - loss: 0.2961 - mae: 0.4433
Epoch 798/800
7/7 0s 7ms/step - loss: 0.3484 - mae: 0.4778
Epoch 799/800
7/7 0s 6ms/step - loss: 0.2947 - mae: 0.4407
Epoch 800/800
7/7 0s 6ms/step - loss: 0.2842 - mae: 0.4274
```

---

## Gráfica de la función de pérdida

```
In [...]
# Gráfico de la función de pérdida (loss)
plt.plot(r.history['loss'], label='Pérdida (MSE)', color='blue')
plt.plot(r.history['mae'], label='Error absoluto medio (MAE)', color='orange')
plt.xlabel('Épocas')
plt.ylabel('Error')
plt.title('Evolución del entrenamiento')
plt.legend()
plt.grid(True)
plt.show()
```

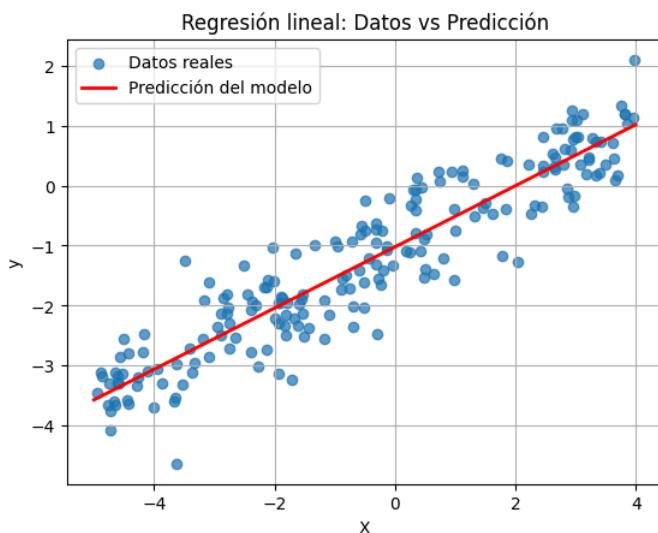


## Predicción

```
In [...]
X_test = np.linspace(-5, 4, 20).reshape(-1, 1)
P_test = modelo.predict(X_test)

1/1 ━━━━━━━━━━ 0s 53ms/step
```

```
In [...]
plt.scatter(X, y, label='Datos reales', alpha=0.7)
plt.plot(X_test, P_test, color='red', label='Predicción del modelo', linewidth=2)
plt.xlabel('X')
plt.ylabel('y')
plt.title('Regresión lineal: Datos vs Predicción')
plt.legend()
plt.grid(True)
plt.show()
```



```
In [...]
# Métricas
modelo.layers[1].get_weights()
```

```
Out[...]: [array([[0.50980544]], dtype=float32), array([-1.0232817], dtype=float32)]
```

## Explicación

Esta línea se usa para **obtener los pesos entrenados** del modelo, específicamente de la **primera capa oculta** (que en este caso es también la única capa densa del modelo).

### 1. `modelo.layers`

Es una lista que contiene todas las **capas del modelo**.

En este ejemplo, el modelo tiene dos capas:

- `layers[0]` → la capa de entrada: `Input(shape=(1,))`
  - `layers[1]` → la capa densa: `Dense(1)`
- 

### 2. `modelo.layers[1]`

Accede específicamente a la **capa densa**, que es donde están los **pesos (w)** y el **sesgo (b)** que se ajustan durante el entrenamiento.

---

### 3. `.get_weights()`

Devuelve una **lista con dos elementos**:

- El primer elemento es un array con el **peso (w)** → forma `(1, 1)` porque es una entrada y una neurona.
  - El segundo elemento es un array con el **sesgo (b)** → forma `(1,)`.
- 

## Ejemplo de salida esperada

Si haces un `print()` a eso después de entrenar el modelo, podrías ver algo como:

```
[array([[0.498]]), array([-0.98])]
```

Esto indicaría que el modelo aprendió aproximadamente la función:

$y \approx 0.498x - 0.98$

muy cercana a la **función original** usada para generar los datos:

$y = 0.5x + \text{ruido}$

---

## Conclusiones

TensorFlow es una biblioteca capaz de implementar redes neuronales para predecir a través de cálculos estadísticos de regresión lineal, cuando un grupo de números, se acerca a los valores reales que se desean predecir. Esta aplicación netamente teórica, puede ser implementada en clasificación de textos para NLP.

---

# 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)

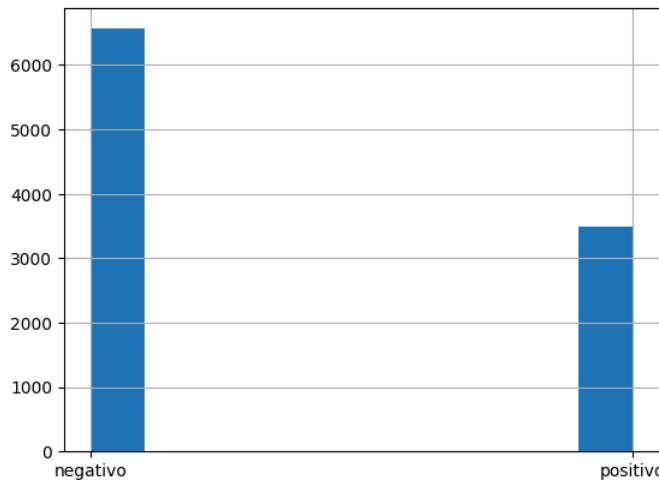
In [...]
df
```

	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	Esta 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 epis...	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
```

	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	Esta 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 epis...	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
)
```

## Predictión

```
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
```

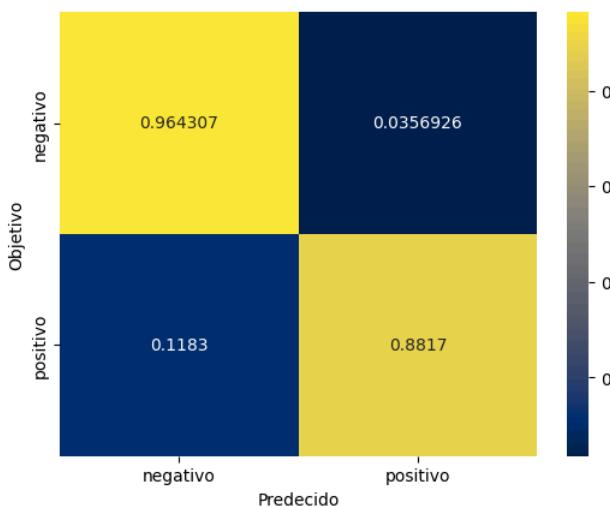
## Matriz de Confusión

```
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 = [  
    'estuve muy entretenida la película',  
    'estuve horrible la película, me aburrió 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 aburrió 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 aburrió mucho',
                 'no la recomiendo']
```

```

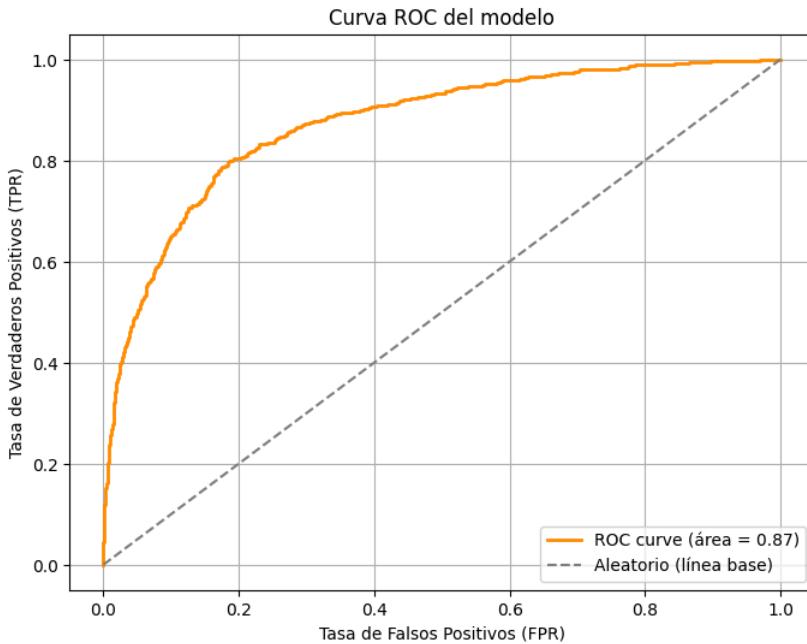
In [...] # Aplicar sigmoid a Logits de test
logits_test = modelo.predict(X_test)
probs_test = 1 / (1 + np.exp(-logits_test)) # sigmoid

# 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  $\approx 0.5 \rightarrow$  modelo no mejor que el azar.
    - AUC  $\approx 1.0 \rightarrow$  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  $\rightarrow$  Tasa de Falsos Positivos.
    - Y  $\rightarrow$  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?

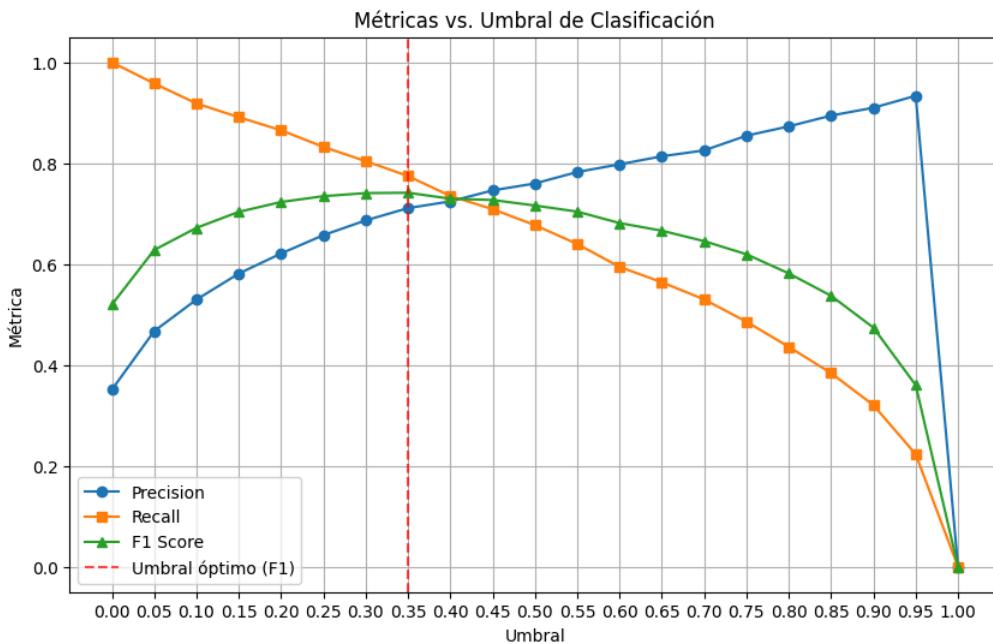
Este procedimiento permite **elegir el umbral óptimo** para convertir las probabilidades a clases, no simplemente usar el típico `0.5`. Así puedes maximizar la métrica que más te interese (en este caso el **F1 Score**, que es útil cuando hay **desequilibrio de clases**).

## 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)', alpha=0.5)

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 = [
    'estuve muy buena la película',
    'estuve horrible la película, me aburrió 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

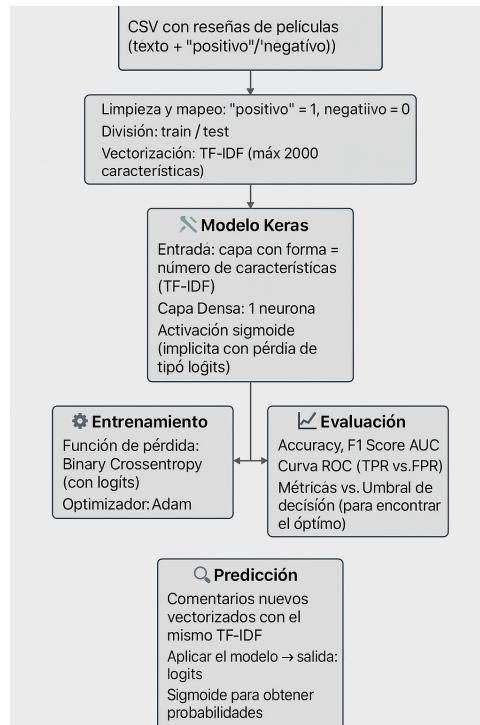
# Interpretar los resultados
for texto, prob in zip(prueba, probabilidades):
    clase = "positivo" if prob >= 0.35 else "negativo"
    print(f'{texto} → {clase} ({prob[0]:.2f})')
```

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.



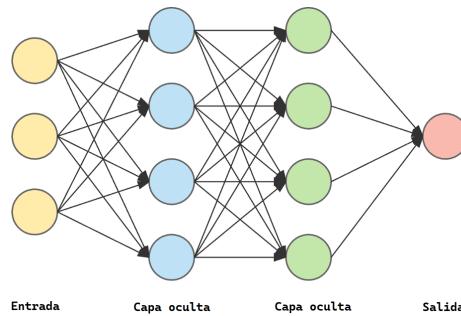
Mg. Luis Felipe Bustamante Narváez

# Anexo 19

## Proyecto 19: Creación de Red Neuronal

Mg. Luis Felipe Bustamante Narváez

En este ejercicio diseñaremos una red neuronal capaz de hacer predicciones sobre diferentes categorías de noticias de un dataset implementado en otros proyectos, donde observaremos como mejora la calidad del proceso con la implementación de las redes.



## Librerías

```
In [...]
import numpy as np
import pandas as pd
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 tensorflow.keras.layers import Dense, Input
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from IPython.display import display, HTML
```

## Cargamos los Datos

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

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

		url	news	Type
0	https://www.larepublica.co/redirect/post/3201905	Durante el foro La banca articulador empresari...		Otra
1	https://www.larepublica.co/redirect/post/3210288	El regulador de valores de China dijo el domin...		Regulaciones
2	https://www.larepublica.co/redirect/post/3240676	En una industria históricamente masculina como...		Alianzas
3	https://www.larepublica.co/redirect/post/3342889	Con el dato de marzo el IPC interanual encaden...		Macroeconomia
4	https://www.larepublica.co/redirect/post/3427208	Ayer en Cartagena se dio inicio a la versión n...		Otra
...	...	...	...	...
1212	https://www.bbva.com/es/como-lograr-que-los-in...	En la vida de toda empresa emergente llega un ...		Innovacion
1213	https://www.bbva.com/es/podcast-como-nos-afect...	La espiral alcista de los precios continúa y g...		Macroeconomia
1214	https://www.larepublica.co/redirect/post/3253735	Las grandes derrotas nacionales son experienci...		Alianzas
1215	https://www.bbva.com/es/bbva-y-barcelona-healt...	BBVA ha alcanzado un acuerdo de colaboración c...		Innovacion
1216	https://www.larepublica.co/redirect/post/3263980	Casi entrando a la parte final de noviembre la...		Alianzas

1217 rows × 3 columns

## Procesamiento de los Datos

### Creamos las categorías

```
In [...] target = df['Type'].astype('category').cat.codes
```

```
In [...] target
```

```
Out[...] 0      3
         1      4
         2      0
         3      2
         4      3
         ..
1212    1
1213    2
1214    0
1215    1
1216    0
Length: 1217, dtype: int8
```

```
In [...] # Adicionamos la columna al df
df['target'] = target
```

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

Out[...]

	url	news	Type	target
0	https://www.larepublica.co/redirect/post/3201905	Durante el foro La banca articulador empresari...	Otra	3
1	https://www.larepublica.co/redirect/post/3210288	El regulador de valores de China dijo el domin...	Regulaciones	4
2	https://www.larepublica.co/redirect/post/3240676	En una industria históricamente masculina como...	Alianzas	0
3	https://www.larepublica.co/redirect/post/3342889	Con el dato de marzo el IPC interanual encaden...	Macroeconomia	2
4	https://www.larepublica.co/redirect/post/3427208	Ayer en Cartagena se dio inicio a la versión n...	Otra	3
...	...	...	...	...
1212	https://www.bbva.com/es/como-lograr-que-los-in...	En la vida de toda empresa emergente llega un ...	Innovacion	1
1213	https://www.bbva.com/es/podcast-como-nos-afect...	La espiral alcista de los precios continúa y g...	Macroeconomia	2
1214	https://www.larepublica.co/redirect/post/3253735	Las grandes derrotas nacionales son experienci...	Alianzas	0
1215	https://www.bbva.com/es/bbva-y-barcelona-healt...	BBVA ha alcanzado un acuerdo de colaboración c...	Innovacion	1
1216	https://www.larepublica.co/redirect/post/3263980	Casi entrando a la parte final de noviembre la...	Alianzas	0

1217 rows × 4 columns

## Separamos el conjunto de Datos

In [...]

```
df_train, df_test = train_test_split(df, test_size=0.3)
```

## Vectorizamos con TF-IDF

In [...]

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

In [...]

```
X_train
```

Out[...]

```
<851x25101 sparse matrix of type '<class 'numpy.float64'>'  
with 203924 stored elements in Compressed Sparse Row format>
```

In [...]

```
X_test
```

Out[...]

```
<366x25101 sparse matrix of type '<class 'numpy.float64'>'  
with 82175 stored elements in Compressed Sparse Row format>
```

## Convertimos los conjuntos a Arreglos (tensorFlow)

In [...]

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

## Creamos los conjuntos de salida

In [...]

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

In [...]

```
len(Y_train)
```

Out[...]

```
851
```

In [...]

```
len(Y_test)
```

Out[...]

```
366
```

## Obtenemos el número de clases

```
In [... K = df['target'].max() + 1  
K
```

```
Out[... 7
```

## Obtenemos las Dimensiones

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

```
In [... D
```

```
Out[... 25101
```

## Explicación

1. `train_test_split(df, test_size=0.3)` divide el DataFrame en un 70% para entrenamiento y un 30% para prueba.
2. `TfidfVectorizer()` convierte el texto (columna `'news'`) en vectores numéricos usando TF-IDF.
3. `fit_transform()` y `transform()` generan la matriz de características para entrenamiento y prueba.
4. `toarray()` convierte las matrices dispersas a arrays NumPy densos.
5. `Y_train` y `Y_test` son las etiquetas objetivo.
6. `K` es el número total de clases (asumiendo clases de 0 a `K-1`).
7. `D` es la dimensión de entrada, es decir, el número de características por muestra.

## Modelo

### Construcción del Modelo

```
In [... # Capa de entrada  
i = Input(shape=(D,))  
# Capa Densa  
x = Dense(300, activation='relu')(i)  
#Capa de salida Softmax  
x = Dense(K)(x)  
  
#Creación del modelo  
modelo = Model(i, x)
```

### Resumen del Modelo

```
In [... modelo.summary()  
Model: "functional"
```

Layer (type)	Output Shape	Param #
input_layer_1 (InputLayer)	(None, 25101)	0
dense_1 (Dense)	(None, 300)	7,530,600
dense_2 (Dense)	(None, 7)	2,107

```
Total params: 7,532,707 (28.73 MB)
Trainable params: 7,532,707 (28.73 MB)
Non-trainable params: 0 (0.00 B)
```

## Compilamos el Modelo

```
In [...] modelo.compile(
    loss= tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
    optimizer='adam',
    metrics=['accuracy']
)
```

## Entrenamos el Modelo

```
In [...] r = modelo.fit(
    X_train,
    Y_train,
    validation_data= (X_test, Y_test),
    epochs= 100,
    batch_size= 12   #Total de datos / 12
)
```

### Explicación

#### 💡 Arquitectura del modelo:

- `Input(shape=(D,))`: entrada de dimensión `D` (viene de la matriz TF-IDF).
- `Dense(300, activation='relu')`: una capa totalmente conectada con 300 neuronas y activación ReLU.
- `Dense(K)`: capa de salida con `K` neuronas (una por clase). No lleva activación porque usamos `from_logits=True` en la pérdida.

#### 💡 Compilación:

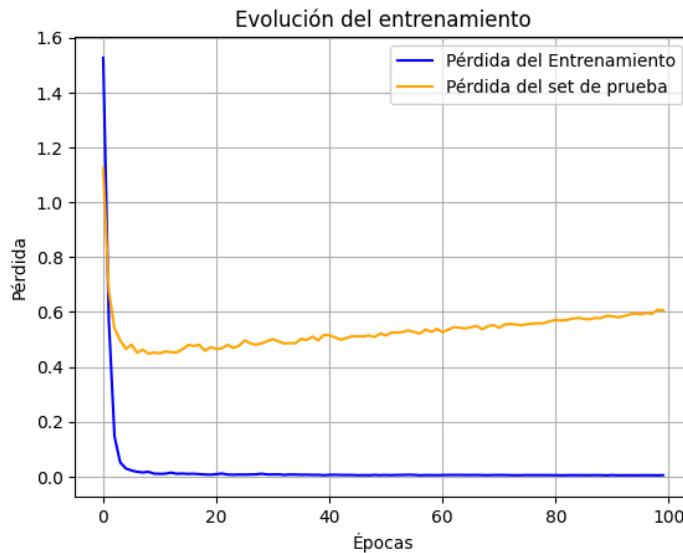
- **Loss:** `SparseCategoricalCrossentropy(from_logits=True)` se usa porque las etiquetas están codificadas como enteros y la salida aún no pasa por softmax.
- **Optimizer:** `'adam'` para una optimización eficiente.
- **Métrica:** `'accuracy'` para evaluar el desempeño.

#### 📊 Entrenamiento:

- Entrena durante `100` épocas, usando lotes de tamaño `12`.
- Se valida en cada época usando `X_test` y `Y_test`.

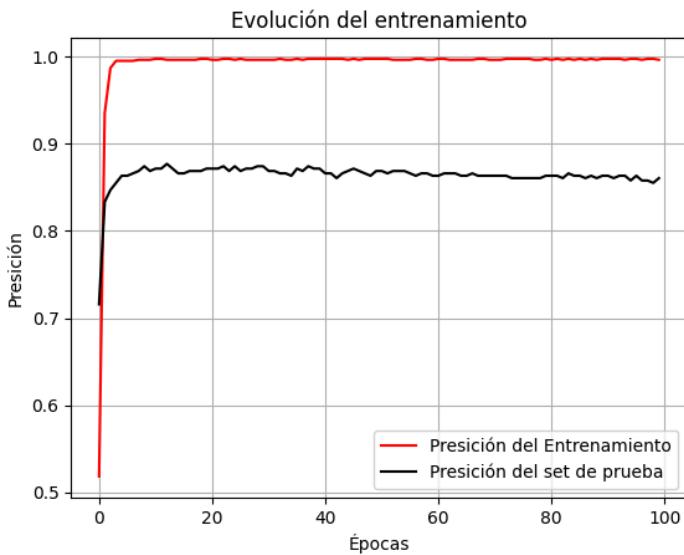
## Gráfico de la pérdida por iteración

```
In [...]
# Gráfico de la función de pérdida (loss)
plt.plot(r.history['loss'], label='Pérdida del Entrenamiento', color='blue')
plt.plot(r.history['val_loss'], label='Pérdida del set de prueba', color='orange')
plt.xlabel('Épocas')
plt.ylabel('Pérdida')
plt.title('Evolución del entrenamiento')
plt.legend()
plt.grid(True)
plt.show()
```



## Gráfico de la presición por iteración

```
In [...]
# Gráfico de la métrica de presición (accuracy)
plt.plot(r.history['accuracy'], label='Presición del Entrenamiento', color='red')
plt.plot(r.history['val_accuracy'], label='Presición del set de prueba', color='black')
plt.xlabel('Épocas')
plt.ylabel('Presición')
plt.title('Evolución del entrenamiento')
plt.legend()
plt.grid(True)
plt.show()
```



## Prueba del Modelo

```
In [...]
# Ejemplo de una nueva noticia
nueva_noticia = ["El presidente anunció nuevas medidas \
                  económicas para enfrentar la inflación."]

# Transformar la noticia con el mismo vectorizador usado en entrenamiento
X_nueva = vectorizer.transform(nueva_noticia).toarray()

# Obtener la predicción (logits)
Predict = modelo.predict(X_nueva)

# Aplicar softmax para convertir P en probabilidades
probs = tf.nn.softmax(Predict).numpy()

# Obtener la clase predicha
clase_predicha = np.argmax(probs)

# Obtener el mapeo de códigos a etiquetas
category_mapping = dict(enumerate(df['Type'].astype('category').cat.categories))

# Creamos diccionario para almacenar la probabilidad de cada categoría
probs_dict = {
    category_mapping[i]: float(prob)
    for i, prob in enumerate(probs[0])
}
display(HTML(f"<h3><b>Clase predicha para la noticia:</b>\n{category_mapping[clase_predicha]}</h3>"))

#Salida en pandas
# Crear DataFrame con categorías y probabilidades
df_probs = pd.DataFrame({
    'Categoría': list(probs_dict.keys()),
    'Probabilidad': list(probs_dict.values())
})

# Ordenar por probabilidad descendente
```

```

df_probs = df_probs.sort_values(by='Probabilidad', ascending=False).reset_index(drop=True)

# Mostrar la tabla
df_probs

```

1/1 ━━━━━━━━ 0s 41ms/step

## Clase predicha para la noticia: Macroeconomía

Out[...]

	Categoría	Probabilidad
0	Macroeconomía	0.919483
1	Sostenibilidad	0.055128
2	Alianzas	0.019083
3	Otra	0.004394
4	Regulaciones	0.001539
5	Reputación	0.000312
6	Innovación	0.000060

## Gráfico de la salida

In [...]

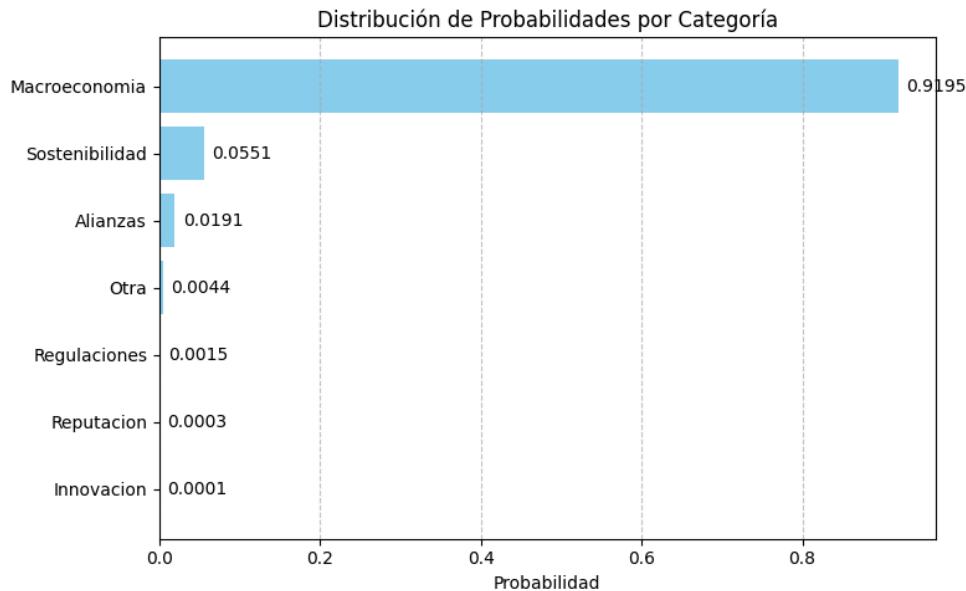
```

# Gráfico de barras
plt.figure(figsize=(8, 5))
bars = plt.barh(df_probs['Categoría'], df_probs['Probabilidad'], color='skyblue')
plt.xlabel('Probabilidad')
plt.title('Distribución de Probabilidades por Categoría')
plt.gca().invert_yaxis() # Opcional: categoría más probable arriba
plt.grid(axis='x', linestyle='--', alpha=0.7)

#Agregamos etiquetas a cada barra
for bar in bars:
    width = bar.get_width()
    plt.text(width + 0.01,
              bar.get_y() + bar.get_height()/2,
              f'{width:.4f}',
              va= 'center'
            )

plt.tight_layout()
plt.show()

```



## Conclusiones

En este modelo, se logró crear una clasificación a través de keras, capaz de identificar, a partir del contenido de una noticia, cuál es su categoría. Con un entrenamiento más robusto, podríamos generar un clasificador de mayor presición capaz de clasificar cualquier texto informativo.

---

Mg. Luis Felipe Bustamante Narváez

# Anexo 20

## Proyecto 20: Manejo de TensorFlow para NLP

Mg. Luis Felipe Bustamante Narváez

En este ejercicio, aplicaremos sentencias de TensorFlow para procesar estructuras de datos, con el fin de prepararnos para siguientes proyectos.

### Librerías

```
In [...]
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
```

### Datos

```
In [...]
oraciones = ['me gusta el fútbol e ir al estadio',
             'juego fútbol los fines de semana.',
             'no me gusta perder']
```

### Procesamiento de Datos

```
In [...]
#Vocabulario máximo
max_vocab_size = 30000
#Iniciamos el tokenizador
tokenizer = Tokenizer(num_words=max_vocab_size)
#Tokenizamos
tokenizer.fit_on_texts(oraciones)
#Creamos las secuencias
secuencias = tokenizer.texts_to_sequences(oraciones)
```

```
In [...]
print(secuencias)
[[1, 2, 4, 3, 5, 6, 7, 8], [9, 3, 10, 11, 12, 13], [14, 1, 2, 15]]
```

```
In [...]
#Diccionario de palabras (Los ordena por peso: mayor frecuencia)
tokenizer.word_index
```

```
Out[...]: {'me': 1,
          'gusta': 2,
          'fútbol': 3,
          'el': 4,
          'e': 5,
          'ir': 6,
          'al': 7,
          'estadio': 8,
          'juego': 9,
          'los': 10,
          'fines': 11,
          'de': 12,
          'semana': 13,
          'no': 14,
          'perder': 15}
```

```
In [...]: #Organizamos la secuencia estandarizando la cantidad de elementos por vector
data = pad_sequences(secuencias)
print(data)

[[ 1  2  4  3  5  6  7  8]
 [ 0  0  9  3 10 11 12 13]
 [ 0  0  0  0 14  1  2 15]]
```

```
In [...]: #Limitamos el número de palabras con relleno al principio
max_secuence_length = 5
data = pad_sequences(secuencias, maxlen=max_secuence_length)
print(data)

[[ 3  5  6  7  8]
 [ 3 10 11 12 13]
 [ 0 14  1  2 15]]
```

```
In [...]: #Limitamos el número de palabras con relleno al final
max_secuence_length = 5
data = pad_sequences(secuencias, maxlen=max_secuence_length, padding='post')
print(data)

[[ 3  5  6  7  8]
 [ 3 10 11 12 13]
 [14  1  2 15  0]]
```

```
In [...]: # cambiamos los valores directamente en el método
data = pad_sequences(secuencias, maxlen=6)
print(data)

[[ 4  3  5  6  7  8]
 [ 9  3 10 11 12 13]
 [ 0  0 14  1  2 15]]
```

# Conclusiones

Este proyecto permite indagar sobre algunas sentencias de TensorFlow para procesar datos, tema que será de gran aporte, cuando desarrollemos proyectos robustos de Inteligencia Artificial.

---

Mg. Luis Felipe Bustamante Narváez

# Anexo 21

## Proyecto 21: Clasificador de Texto con CNN's para NLP

Mg. Luis Felipe Bustamante Narváez

En este proyecto, diseñaremos un clasificador de texto, utilizando redes neuronales convolucionales, recurso que permite un procesamiento más complejo de los ejercicios anteriores, pero a su vez más preciso y con excelente eficiencia, y mínimo coste computacional.

### Librerías

```
In [...]
import numpy as np
import pandas as pd
import tensorflow as tf
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.layers import Dense, Input, GlobalMaxPooling1D
from tensorflow.keras.layers import Conv1D, MaxPooling1D, Embedding
from tensorflow.keras.models import Model
from tensorflow.keras.losses import SparseCategoricalCrossentropy
import itertools
from keras.models import load_model
from keras.optimizers import RMSprop
```

### Cargamos los Datos

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

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

Out[...]

		url	news	Type
0	https://www.larepublica.co/redirect/post/3201905	Durante el foro La banca articulador empresari...		Otra
1	https://www.larepublica.co/redirect/post/3210288	El regulador de valores de China dijo el domin...		Regulaciones
2	https://www.larepublica.co/redirect/post/3240676	En una industria históricamente masculina como...		Alianzas
3	https://www.larepublica.co/redirect/post/3342889	Con el dato de marzo el IPC interanual encaden...	Macroeconomia	
4	https://www.larepublica.co/redirect/post/3427208	Ayer en Cartagena se dio inicio a la versión n...		Otra
...	...	...	...	...
1212	https://www.bbva.com/es/como-lograr-que-los-in...	En la vida de toda empresa emergente llega un ...		Innovacion
1213	https://www.bbva.com/es/podcast-como-nos-afect...	La espiral alcista de los precios continúa y g...	Macroeconomia	
1214	https://www.larepublica.co/redirect/post/3253735	Las grandes derrotas nacionales son experienci...		Alianzas
1215	https://www.bbva.com/es/bbva-y-barcelona-healt...	BBVA ha alcanzado un acuerdo de colaboración c...		Innovacion
1216	https://www.larepublica.co/redirect/post/3263980	Casi entrando a la parte final de noviembre la...		Alianzas

1217 rows × 3 columns

## Procesamiento de Datos

### Creamos las categorías

```
In [...] target = df['Type'].astype('category').cat.codes
```

```
In [...] target
```

Out[...]

0	3
1	4
2	0
3	2
4	3
...	..
1212	1
1213	2
1214	0
1215	1
1216	0

Length: 1217, dtype: int8

```
In [...] # Adicionamos la columna al df
df['target'] = target
```

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

	url	news	Type	target
0	https://www.larepublica.co/redirect/post/3201905	Durante el foro La banca articulador empresari...	Otra	3
1	https://www.larepublica.co/redirect/post/3210288	El regulador de valores de China dijo el domin...	Regulaciones	4
2	https://www.larepublica.co/redirect/post/3240676	En una industria históricamente masculina como...	Alianzas	0
3	https://www.larepublica.co/redirect/post/3342889	Con el dato de marzo el IPC interanual encaden...	Macroeconomia	2
4	https://www.larepublica.co/redirect/post/3427208	Ayer en Cartagena se dio inicio a la versión n...	Otra	3
...	...	...	...	...
1212	https://www.bbva.com/es/como-lograr-que-los-in...	En la vida de toda empresa emergente llega un ...	Innovacion	1
1213	https://www.bbva.com/es/podcast-como-nos-afect...	La espiral alcista de los precios continúa y g...	Macroeconomia	2
1214	https://www.larepublica.co/redirect/post/3253735	Las grandes derrotas nacionales son experienci...	Alianzas	0
1215	https://www.bbva.com/es/bbva-y-barcelona-healt...	BBVA ha alcanzado un acuerdo de colaboración c...	Innovacion	1
1216	https://www.larepublica.co/redirect/post/3263980	Casi entrando a la parte final de noviembre la...	Alianzas	0

1217 rows × 4 columns

## Separamos los conjuntos de Datos

```
In [...] df_train, df_test = train_test_split(df, test_size=0.3)
```

## Obtenemos el número de clases

```
In [...] K = df['target'].max() + 1
K
```

Out[...]: 7

## Creamos los conjuntos de salida

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

```
In [...] len(Y_train)
```

Out[...]: 851

```
In [...] len(Y_test)
```

Out[...]: 366

## Tokenización

### Tokenizamos oraciones en Secuencias

```
In [...] #Vocabulario máximo
max_vocab_size = 30000
#Iniciamos el tokenizador
tokenizer = Tokenizer(num_words=max_vocab_size)
```

```
#Tokenizamos
tokenizer.fit_on_texts(df_train['news'])
#Creamos las secuencias
secuencias_train = tokenizer.texts_to_sequences(df_train['news'])
secuencias_test = tokenizer.texts_to_sequences(df_test['news'])
```

## Diccionario de palabras tokenizadas

```
In [...]
# Creamos el diccionario
word2index = tokenizer.word_index
# Calculamos el tamaño del tokenizado
V = len(word2index)
# mostramos
print(f'Se encontraron {V} tokens.')
```

Se encontraron 26638 tokens.

```
In [...]
diez = dict(itertools.islice(word2index.items(), 10))
print(f'Estas son las 10 primeras palabras que más se repiten son:\n{diez}')

Estas son las 10 primeras palabras que más se repiten son:
{'de': 1, 'la': 2, 'en': 3, 'el': 4, 'que': 5, 'y': 6, 'a': 7, 'los': 8, 'la
s': 9, 'del': 10}
```

## Rellenamos las Secuencias (padding)

```
In [...]
# Rellenar la secuencia de entrenamiento
data_train = pad_sequences(secuencias_train)
print(f'Dimensiones del tensor de entrenamiento: {data_train.shape}')
# Longitud de la secuencia de entrenamiento
T = data_train.shape[1]
print(f'Longitud de la secuencia de entrenamiento: {T}'')
```

Dimensiones del tensor de entrenamiento: (851, 3015)

Longitud de la secuencia de entrenamiento: 3015

```
In [...]
# Rellenar la secuencia de prueba
data_test = pad_sequences(secuencias_test, maxlen=T)
print(f'Dimensiones del tensor de prueba: {data_test.shape}')
# Longitud de la secuencia de prueba
print(f'Longitud de la secuencia de prueba: {data_test.shape[1]}')
```

Dimensiones del tensor de prueba: (366, 3015)

Longitud de la secuencia de prueba: 3015

## Embedding y Modelo

### Dimensiones del Embedding

```
In [... # A modo de prueba  
D = 50
```

## Construcción del Modelo

```
In [... # Capa de entrada  
i = Input(shape=(T,))  
# Capa de embedding  
x = Embedding(V + 1, D)(i) #+1 para el token especial de palabras desconocida  
# Capa de convolución  
x = Conv1D(32, 3, activation='relu')(x) # 32 filtros por cada 3 palabras  
# Capa de pooling  
x = GlobalMaxPooling1D()(x)  
  
# Otras capas que podemos aplicar según lo visto  
#x = MaxPooling1D(3)(x)  
#x = Conv1D(64, 3, activation='relu')(x)  
#x = MaxPooling1D(3)(x)  
#x = Conv1D(128, 3, activation='relu')(x)  
  
# Capa Densa  
x = Dense(K)(x)  
  
# Creación del modelo  
modelo = Model(i, x)
```

## Resumen del Modelo

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

Model: "functional"

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 3015)	0
embedding (Embedding)	(None, 3015, 50)	1,331,950
conv1d (Conv1D)	(None, 3013, 32)	4,832
global_max_pooling1d (GlobalMaxPooling1D)	(None, 32)	0
dense (Dense)	(None, 7)	231

Total params: 1,337,013 (5.10 MB)

Trainable params: 1,337,013 (5.10 MB)

Non-trainable params: 0 (0.00 B)

## Compilamos el Modelo

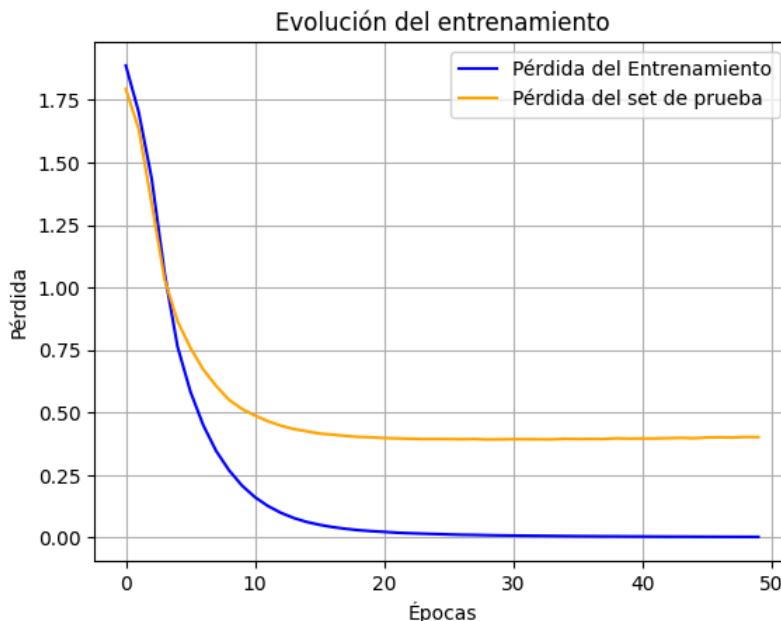
```
In [...]
    modelo.compile(
        loss= SparseCategoricalCrossentropy(from_logits=True),
        optimizer='adam',
        metrics=['accuracy']
    )
```

## Entrenamos el Modelo

```
In [...]
print('Entrenando el modelo...')
r = modelo.fit(
    data_train,
    Y_train,
    epochs=50,
    validation_data=(data_test, Y_test)
)
```

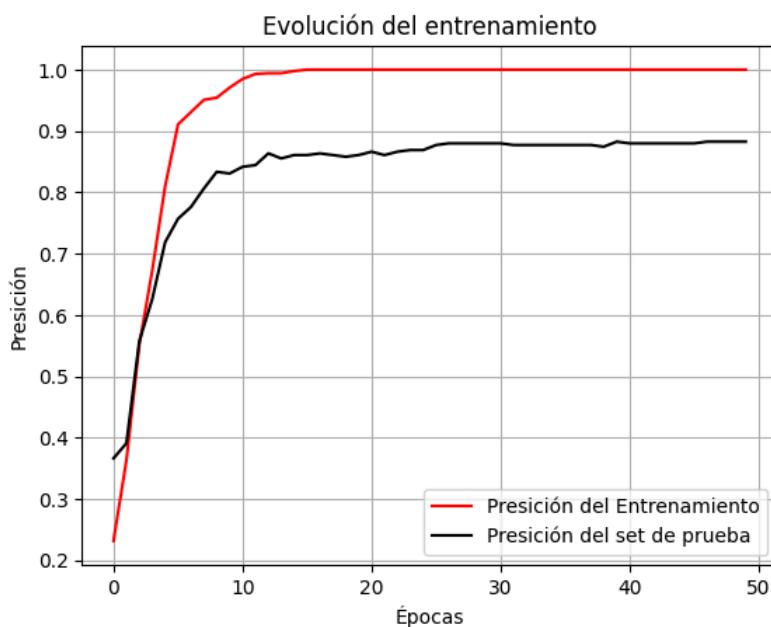
## Gráfico de la pérdida por iteración

```
In [...]
# Gráfico de La función de pérdida (Loss)
plt.plot(r.history['loss'], label='Pérdida del Entrenamiento', color='blue')
plt.plot(r.history['val_loss'], label='Pérdida del set de prueba', color='orange')
plt.xlabel('Épocas')
plt.ylabel('Pérdida')
plt.title('Evolución del entrenamiento')
plt.legend()
plt.grid(True)
plt.show()
```



## Gráfico de la presición por iteración

```
In [...]
# Gráfico de la métrica de presición (accuracy)
plt.plot(r.history['accuracy'], label='Presición del Entrenamiento', color='red')
plt.plot(r.history['val_accuracy'], label='Presición del set de prueba', color='black')
plt.xlabel('Épocas')
plt.ylabel('Presición')
plt.title('Evolución del entrenamiento')
plt.legend()
plt.grid(True)
plt.show()
```



## Guardar Modelo y sus Pesos

```
In [...]
# Archivo con extensión HDF5 (depreciado) o keras (actual)
modelo.save('modelo21.keras')
print('Modelo guardado con éxito.')
Modelo guardado con éxito.
```

```
In [...]
# Guardamos los pesos
modelo.save_weights('modelo21_pesos.weights.h5')
print('Pesos del modelo guardados con éxito.')
Pesos del modelo guardados con éxito.
```

## Cargar el Modelo para futuras pruebas

```
In [...]
model_load = load_model('modelo21.keras', compile=False)
model_load.compile()
```

```
    optimizer = RMSprop(),
    loss = SparseCategoricalCrossentropy(from_logits=True),
    metrics = ['accuracy']
)

print(f'El Modelo {model_load} se ha cargado, y recompilado correctamente.')
El Modelo <Functional name=functional, built=True> se ha cargado, y recompila
do correctamente.
```

## Probamos el Modelo con Datos nuevos

### Función para predecir texto

```
In [...]
def predecir_texto(texto, modelo, tokenizer, T, idx2label=None):
    # Asegurarse que el texto está en una lista
    if isinstance(texto, str):
        texto = [texto]

    # Tokenizar y hacer padding
    secuencia = tokenizer.texts_to_sequences(texto)
    secuencia_padded = pad_sequences(secuencia, maxlen=T)

    # Predicción
    pred = modelo.predict(secuencia_padded)
    clase_predicha = np.argmax(pred, axis=1)[0]

    # Mostrar resultado
    if idx2label:
        print(f'Clase predicha: {clase_predicha} ({idx2label[clase_predicha]})')
        return idx2label[clase_predicha]
    else:
        print(f'Clase predicha: {clase_predicha}')
        return clase_predicha
```

### Llamamos la función

```
In [...]
texto_de_prueba = "Este es un ejemplo de noticia económica internacional"
# Crear mapeo inverso de índices a nombres
idx2label = dict(enumerate(df['Type'].astype('category').cat.categories))
predecir_texto(texto_de_prueba, model_load, tokenizer, T)
idx2label

1/1 ━━━━━━━━━━ 0s 102ms/step
Clase predicha: 2
```

```
Out[...]: {0: 'Alianzas',
          1: 'Innovacion',
          2: 'Macroeconomia',
          3: 'Otra',
          4: 'Regulaciones',
          5: 'Reputacion',
          6: 'Sostenibilidad'}
```

## Conclusiones

En este modelo, se logró crear una clasificación a través de keras, capaz de identificar, a partir del contenido de una noticia, cuál es su categoría. Con un entrenamiento a través de embeddings y redes neuronales convolucionales, hemos generado un clasificador de mayor presición capaz de clasificar cualquier texto informativo.

---

Mg. Luis Felipe Bustamante Narváez

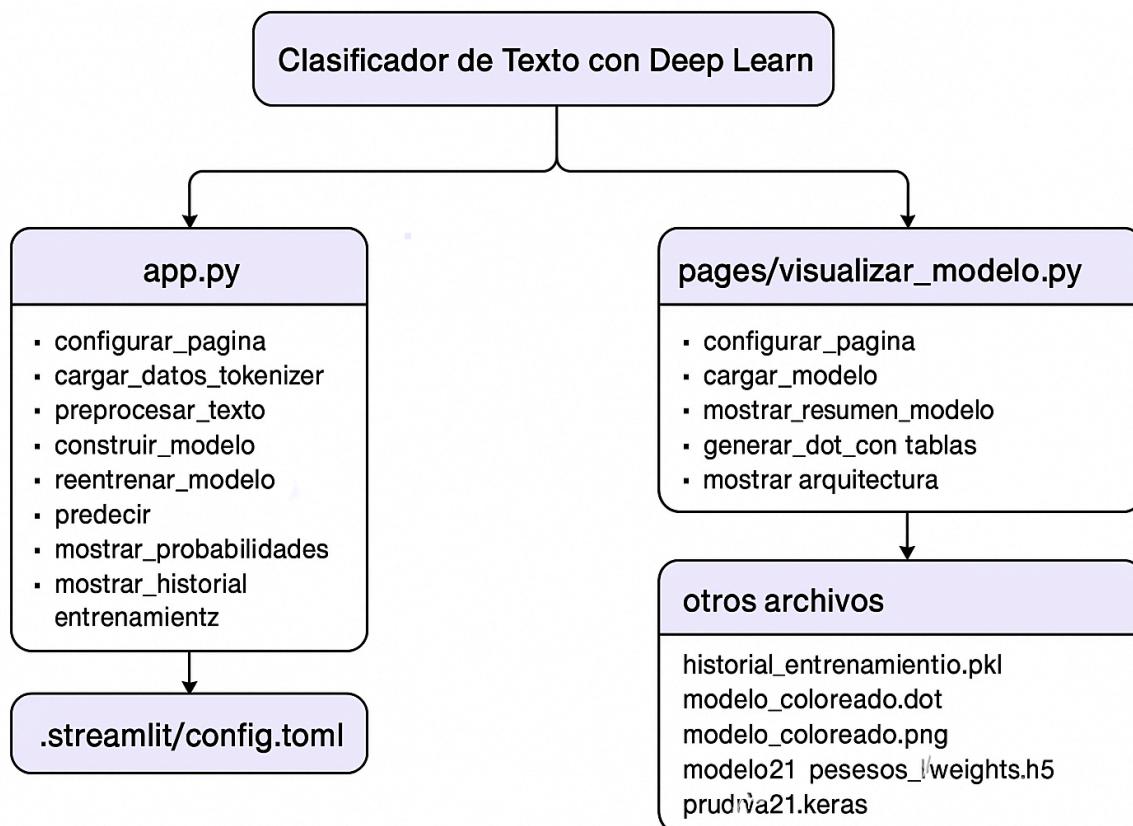
## Anexo 22

### Proyecto 22: Clasificador de Texto en Streamlit

Mg. Luis Felipe Bustamante Narváez

Este proyecto es la continuación del [Proyecto 21](#), en el cuál diseñamos el modelo con [CNN's](#) y para el caso, lo implementamos en [Streamlit](#), a partir de la generación del archivo [.py](#), para su correcta ejecución y despliegue, el cual se encuentra en el enlace:

[!\[\]\(647e44ea77c89a016b9d36ad68afc84b\_img.jpg\) \*\*cnn-classicator-app.streamlit.app\*\*](#)



### Aplicación Principal ([app.py](#))

```
In [...]
# === Imports y Configuración Inicial ===
import streamlit as st
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import pickle
import os
import tensorflow as tf
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
from sklearn.model_selection import train_test_split
```

```

from tensorflow.keras.models import load_model, Model
from tensorflow.keras.layers import Input, Embedding, Conv1D, GlobalMaxPooling1D, Dense
from tensorflow.keras.losses import SparseCategoricalCrossentropy
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.preprocessing.sequence import pad_sequences

# === Configuración de Streamlit ===
def configurar_pagina():
    st.set_page_config(page_title="Clasificador de Texto", layout="wide")
    st.markdown("""
        <style>
            .main .block-container { max-width: 95%;
            padding-left: 3rem;
            padding-right: 3rem; }
            pre { white-space: pre-wrap !important;
            word-break: break-word !important; }
        </style>
    """, unsafe_allow_html=True)
    st.title("⭐ Clasificador de Texto con Deep Learning")

# === Carga de Datos y Tokenizer ===
def cargar_datos_tokenizer():
    df = pd.read_csv('data/df_total.csv')
    df['target'] = df['Type'].astype('category').cat.codes
    idx2label = dict(enumerate(df['Type'].astype('category').cat.categories))
    label2idx = {v: k for k, v in idx2label.items()}

    if os.path.exists('data/new_examples.csv'):
        df_new = pd.read_csv('data/new_examples.csv')
        df_new['target'] = df_new['Type'].map(label2idx)
        df = pd.concat([df, df_new], ignore_index=True)

    with open('tokenizer.pkl', 'rb') as f:
        tokenizer = pickle.load(f)

    return df, tokenizer, idx2label, label2idx

# === Preprocesamiento de Texto ===
def preprocesar_texto(df, tokenizer):
    sequences = tokenizer.texts_to_sequences(df['news'])
    data = pad_sequences(sequences)
    T = data.shape[1]
    return data, T

# === Construcción del Modelo ===
def construir_modelo(V, T, K, D=50):
    i = Input(shape=(T,))
    x = Embedding(V + 1, D)(i)
    x = Conv1D(32, 3, activation='relu')(x)
    x = GlobalMaxPooling1D()(x)
    x = Dense(K)(x)
    modelo = Model(i, x)
    modelo.compile(loss=SparseCategoricalCrossentropy(from_logits=True),

```

```

        optimizer=Adam(),
        metrics=['accuracy'])
    return modelo

# === Reentrenamiento del Modelo ===
def reentrenar_modelo(df, data, T, label2idx, tokenizer):
    st.sidebar.success("🕒 Entrenando modelo...")
    V = len(tokenizer.word_index)
    K = len(label2idx)

    modelo = construir_modelo(V, T, K)
    X_train, X_test, y_train, y_test = train_test_split(data,
                                                        df['target'].values,
                                                        test_size=0.3,
                                                        random_state=42)
    history = modelo.fit(X_train, y_train, epochs=10,
                          validation_data=(X_test, y_test))

    modelo.save('modelo21.keras')
    with open('historial_entrenamiento.pkl', 'wb') as f:
        pickle.dump(history.history, f)

    if os.path.exists('data/new_examples.csv'):
        os.remove('data/new_examples.csv')

    st.sidebar.success("✅ Reentrenamiento completo")
    st.session_state.retrain = False

# === Predicción ===
def predecir(modelo, tokenizer, T, texto):
    seq = tokenizer.texts_to_sequences([texto])
    padded = pad_sequences(seq, maxlen=T)
    pred = modelo.predict(padded)
    probs = tf.nn.softmax(pred[0]).numpy()
    return probs

# === Visualización de Probabilidades ===
def mostrar_probabilidades(probs, idx2label):
    df_probs = pd.DataFrame({
        'Clase': list(idx2label.values()),
        'Probabilidad': probs
    }).sort_values('Probabilidad', ascending=False)
    st.dataframe(df_probs, use_container_width=True)

    fig, ax = plt.subplots()
    colors = plt.cm.tab20.colors[:len(df_probs)]
    bar_colors = [colors[list(idx2label.values()).index(clase)]] \
        for clase in df_probs['Clase']]
    ax.barh(df_probs['Clase'], df_probs['Probabilidad'], color=bar_colors)
    ax.invert_yaxis()
    ax.grid(True, axis='x')
    st.pyplot(fig)

```

```

# === Visualización de Entrenamiento ===
def mostrar_historial_entrenamiento():
    try:
        with open('historial_entrenamiento.pkl', 'rb') as f:
            history = pickle.load(f)

            fig1, ax1 = plt.subplots()
            ax1.plot(history['loss'], label='Pérdida (entrenamiento)', color='blue')
            ax1.plot(history['val_loss'], label='Pérdida (validación)', color='orange')
            ax1.legend(); ax1.grid(); ax1.set_title("Pérdida"); st.pyplot(fig1)

            if 'accuracy' in history:
                fig2, ax2 = plt.subplots()
                ax2.plot(history['accuracy'], label='Precisión (entrenamiento)',
                          color='red')
                ax2.plot(history['val_accuracy'], label='Precisión (validación)',
                          color='black')
                ax2.legend(); ax2.grid(); ax2.set_title("Precisión"); st.pyplot(fig2)

    except FileNotFoundError:
        st.info("⚠️ No se encontró el archivo `historial_entrenamiento.pkl`.")


# === Matriz de Confusión ===
def mostrar_matriz_confusion(df, modelo, tokenizer, T, idx2label):
    try:
        _, df_test = train_test_split(df, test_size=0.3, random_state=42)
        X_test = pad_sequences(tokenizer.texts_to_sequences(df_test['news']),
                               maxlen=T)
        y_test = df_test['target'].values

        y_pred = np.argmax(modelo.predict(X_test), axis=1)
        cm = confusion_matrix(y_test, y_pred)
        disp = ConfusionMatrixDisplay(confusion_matrix=cm,
                                       display_labels=list(idx2label.values()))

        fig, ax = plt.subplots(figsize=(10, 8))
        disp.plot(ax=ax, cmap='Blues', xticks_rotation=45, colorbar=False)
        ax.set_title("Matriz de Confusión")
        st.pyplot(fig)
    except Exception as e:
        st.warning(f"⚠️ Error al generar la matriz de confusión: {e}")


# === Distribución de Clases ===
def mostrar_distribucion_clases(df):
    fig, ax = plt.subplots()
    counts = df['Type'].value_counts()
    colors = plt.cm.tab20.colors[:len(counts)]
    counts.plot(kind='bar', color=colors, ax=ax)
    ax.set_title('Cantidad de muestras por clase')
    st.pyplot(fig)


# === Evaluación del Modelo ===
def evaluar_modelo(df, modelo, tokenizer, T):
    try:

```

```

_, df_test = train_test_split(df, test_size=0.3, random_state=42)
X_test = pad_sequences(tokenizer.texts_to_sequences(df_test['news']), maxlen=T)
y_test = df_test['target'].values

loss, acc = modelo.evaluate(X_test, y_test, verbose=0)
st.success(f" ✅ Precisión del modelo: **{acc:.4f}**")
st.info(f" 📈 Pérdida del modelo: **{loss:.4f}**")
except Exception as e:
    st.warning(f"⚠️ No se pudo evaluar el modelo: {e}")

# pie de página
def mostrar_firma_sidebar():
    st.sidebar.markdown("""
        <style>
            .firma-sidebar {
                position: fixed;
                bottom: 20px;
                left: 0;
                width: 20%;
                padding: 10px 15px;
                font-size: 0.8rem;
                border-radius: 10px;
                background-color: rgba(250, 250, 250, 0.9);
                z-index: 9999;
                text-align: left;
            }
            .firma-sidebar a {
                text-decoration: none;
                color: #333;
            }
            .firma-sidebar a:hover {
                color: #0077b5;
            }
        </style>

        <div class="firma-sidebar">
            Desarrollado por <strong>Mg. Luis Felipe Bustamante Narváez</strong><br>
            <a href="https://github.com/luizbn2" target="_blank">/github</a> .
            <a href="https://www.linkedin.com/in/lfbn2" target="_blank">/LinkedIn</a>
        </div>
    """", unsafe_allow_html=True)

# === Interfaz Principal ===
def main():
    configurar_pagina()

    with st.sidebar:
        st.header("⚙️ Redes Neuronales")
        st.page_link("pages/visualizar_modelo.py", label="✳️ Estructura CNN")

        st.header("⚙️ Opciones Avanzadas")
        if st.button("🔁 Reentrenar modelo"):
            st.session_state.retrain = True

```

```

st.info("🕒 Cargando datos y modelo...")
df, tokenizer, idx2label, label2idx = cargar_datos_tokenizer()
data, T = preprocesar_texto(df, tokenizer)

if st.session_state.get("retrain"):
    reentrenar_modelo(df, data, T, label2idx, tokenizer)

modelo = load_model('modelo21.keras', compile=False)
modelo.compile(optimizer='adam',
                loss=SparseCategoricalCrossentropy(from_logits=True),
                metrics=['accuracy'])

# Entrada
st.subheader("🌐 Clasificación de texto")
metodo = st.radio("Selecciona el método de entrada:",
                  ('Escribir texto', 'Subir archivo .txt'))
texto = st.text_area("Texto a clasificar:") if metodo == 'Escribir texto' else \
(st.file_uploader("Sube archivo .txt", type="txt").read().decode("utf-8") \
if st.file_uploader("Sube archivo .txt", type="txt") else "")

if texto:
    probs = predecir(modelo, tokenizer, T, texto)
    pred_idx = np.argmax(probs)
    pred_label = idx2label[pred_idx]
    st.success(f"🔍 Predicción: **{pred_label}**")

    mostrar_probabilidades(probs, idx2label)

    st.subheader("📝 Corrección de etiqueta")
    correc_label = st.selectbox("Categoría correcta (si aplica):",
                                list(idx2label.values()))
    if st.button("✅ Confirmar y aprender"):
        nuevo_df = pd.DataFrame({'news': [texto], 'Type': [correc_label]})
        nuevo_df['target'] = nuevo_df['Type'].map(label2idx)
        if os.path.exists('data/new_examples.csv'):
            df_ant = pd.read_csv('data/new_examples.csv')
            nuevo_df = pd.concat([df_ant, nuevo_df], ignore_index=True)
        nuevo_df.to_csv('data/new_examples.csv', index=False)
        st.success("🧠 Guardado para reentrenamiento")

    st.markdown("---")
    st.subheader("📈 Evolución del entrenamiento")
    mostrar_historial_entrenamiento()

    st.markdown("---")
    st.subheader("📊 Matriz de Confusión")
    mostrar_matriz_confusion(df, modelo, tokenizer, T, idx2label)

    st.markdown("---")
    st.subheader("📊 Distribución de Clases")
    mostrar_distribucion_clases(df)

    st.markdown("---")
    st.subheader("📌 Evaluación final del modelo")
    evaluar_modelo(df, modelo, tokenizer, T)

```

```
# === Lanzar App ===
if __name__ == '__main__':
    main()
    mostrar_firma_sidebar()
```

## Explicación

---

### Importación de librerías y configuración inicial

Se cargan librerías clave para el funcionamiento del proyecto:

- `streamlit`, para crear interfaces interactivas.
  - `numpy` y `pandas`, para manejar datos.
  - `matplotlib`, para gráficos.
  - `tensorflow.keras` y `sklearn`, para el modelado de redes neuronales y evaluación del desempeño.
- 

### Configuración de la interfaz Streamlit

La función `configurar_pagina()` define el título, el ancho de página y algunos estilos personalizados con HTML y CSS para mejorar la visualización.

---

### Carga de datos y tokenizer

En `cargar_datos_tokenizer()`, se carga un CSV con textos ya etiquetados y se convierten las etiquetas a valores numéricos con Pandas. Luego:

- Si hay nuevos ejemplos para aprender, se agregan al DataFrame.
  - Se carga un tokenizer guardado previamente con `pickle`, que transforma texto en secuencias de números.
- 

### Preprocesamiento de texto

La función `preprocesar_texto(df, tokenizer)` convierte los textos en secuencias numéricas y las ajusta a una longitud uniforme usando `pad_sequences`.

---

### Definición del modelo CNN

La función `construir_modelo(V, T, K, D=50)` construye una red neuronal basada en una arquitectura CNN sencilla:

- `Embedding`, convierte índices en vectores.
- `Conv1D`, extrae patrones locales.
- `GlobalMaxPooling1D`, reduce la dimensionalidad.
- `Dense`, genera predicciones por clase.

Se compila con:

- `SparseCategoricalCrossentropy`, para clasificación multiclase.
  - `Adam`, como optimizador.
- 

## Reentrenamiento del modelo

La función `reentrenar_modelo()` entrena desde cero:

- Usa 70% entrenamiento y 30% prueba.
  - Guarda el modelo como `modelo21.keras`.
  - Guarda el historial de entrenamiento.
  - Elimina los nuevos ejemplos ya utilizados, si existen.
- 

## Predicción del texto

La función `predecir(modelo, tokenizer, T, texto)`:

- Preprocesa un texto nuevo.
  - Usa el modelo cargado para predecir.
  - Devuelve las probabilidades por clase.
- 

## Visualización de resultados

### 1. `mostrar_probabilidades()`

- Muestra en tabla y gráfico de barras las probabilidades de cada clase para una predicción.

### 2. `mostrar_historial_entrenamiento()`

- Si existe el historial de entrenamiento guardado, genera gráficos de pérdida y precisión para evaluar el rendimiento.

### 3. `mostrar_matriz_confusion()`

- Muestra visualmente cuántas predicciones fueron correctas o incorrectas por clase.

### 4. `mostrar_distribucion_clases()`

- Visualiza cuántos ejemplos hay por clase, útil para detectar desbalances en los datos.

### 5. `evaluar_modelo()`

- Calcula la pérdida y precisión del modelo con el conjunto de prueba.
- 

## Firma personalizada en el sidebar

La función `mostrar_firma_sidebar()` agrega un pie de firma personalizado con links a GitHub y LinkedIn del autor.

---

## Función principal

`main()` organiza la aplicación:

- Carga el modelo y los datos.
  - Ofrece reentrenar si el usuario lo solicita.
  - Permite al usuario escribir o subir texto para clasificar.
  - Muestra predicción, visualizaciones, evaluación, y opción para guardar nuevos ejemplos para reentrenar.
- 

## Ejecución de la app

Finalmente, el bloque:

```
if __name__ == '__main__'
```

llama a `main()` y luego a `mostrar_firma_sidebar()`, ejecutando toda la app al correr el script.

---

## Visualizador del Modelo (`visualizar_modelo.py`)

```
In [...]
# === Imports ===
import streamlit as st
from tensorflow.keras.models import load_model
import matplotlib.pyplot as plt
import os
import io
from PIL import Image
from contextlib import redirect_stdout
import pydot

# === Configuración de Página ===
def configurar_pagina():
    st.set_page_config(page_title="Capas del Modelo", layout="wide")
    st.markdown("""
        <style>
            .main .block-container {
                max-width: 95%;
                padding-left: 3rem;
                padding-right: 3rem;
            }
            pre {
                white-space: pre-wrap !important;
                word-break: break-word !important;
            }
        </style>
    """, unsafe_allow_html=True)
    st.sidebar.page_link("app.py", label="🏠 Página Principal")
    st.title("⚡ Visualización de la Red Neuronal")

# === Cargar Modelo ===
def cargar_modelo(ruta):
    st.info("🕒 Cargando modelo...")
```

```

    return load_model(ruta, compile=False)

# === Mostrar Resumen del Modelo ===
def mostrar_resumen_modelo(modelo):
    st.subheader("📋 Resumen del Modelo")
    summary_buffer = io.StringIO()
    with redirect_stdout(summary_buffer):
        modelo.summary()
    st.code(summary_buffer.getvalue(), language='text')

# === Generar archivo DOT personalizado ===
def generar_dot_con_tablas(model, output_path):
    layer_colors = {
        "Embedding": "#dcedc8",
        "Conv1D": "#ffccbc",
        "GlobalMaxPooling1D": "#ffe082",
        "Dense": "#bbdefb",
        "InputLayer": "#f0f0f0"
    }

    def get_shape_safe(tensor):
        try:
            return str(tensor.shape)
        except:
            return "?"

    with open(output_path, "w") as f:
        f.write("digraph G {\n")
        f.write("    rankdir=TB;\n")
        f.write("    concentrate=true;\n")
        f.write("    dpi=200;\n")
        f.write("    splines=ortho;\n")
        f.write("    node [shape=plaintext fontname=Helvetica];\n\n")

        for i, layer in enumerate(model.layers):
            name = layer.name
            tipo = layer.__class__.__name__
            node_id = f"layer_{i}"

            try:
                input_shape = str(layer.input_shape)
            except:
                input_shape = get_shape_safe(layer.input) \
                    if hasattr(layer, "input") else "?"

            try:
                output_shape = str(layer.output_shape)
            except:
                output_shape = get_shape_safe(layer.output) \
                    if hasattr(layer, "output") else "?"

            color = layer_colors.get(tipo, "#eeeeee")

            label = f"""<TABLE BORDER="0" CELLBORDER="1" CELLSPACING="0" CELLPADDING="6" BGCOLOR="{color}">
<TR>
<TD>{name}</TD>
<TD>{tipo}</TD>
<TD>{input_shape}</TD>
<TD>{output_shape}</TD>
</TR>
</TABLE>"""

            f.write(f"    {node_id} [label={label}];\n")
            f.write(f"    {node_id} --> {node_id}_in [label=''];\n")
            f.write(f"    {node_id}_in --> {node_id};\n")
            f.write(f"    {node_id} --> {node_id}_out [label=''];\n")
            f.write(f"    {node_id}_out --> {node_id};\n\n")

    f.write("}")

```

```

<TR><TD COLSPAN="2"><B>{name}</B> ({tipo})</TD></TR>
<TR><TD><FONT POINT-SIZE="10">Input</FONT></TD><TD>
<FONT POINT-SIZE="10">{input_shape}</FONT></TD></TR>
<TR><TD><FONT POINT-SIZE="10">Output</FONT></TD>
<TD><FONT POINT-SIZE="10">{output_shape}</FONT></TD></TR>
</TABLE>>"""
f.write(f'    {node_id} [label={label}];\n')

for i in range(1, len(model.layers)):
    f.write(f'    layer_{i-1} -> layer_{i};\n')

f.write("}\n")

# === Mostrar Visualización de La Arquitectura ===
def mostrar_arquitectura(modelo, dot_output_path, png_output_path):
    st.subheader("🌟 Arquitectura Visual")

    if not os.path.exists(png_output_path):
        try:
            generar_dot_con_tablas(modelo, dot_output_path)
            (graph,) = pydot.graph_from_dot_file(dot_output_path)
            graph.write_png(png_output_path)
        except Exception as e:
            st.warning(f"⚠️ No se pudo generar el diagrama: {e}")

    if os.path.exists(png_output_path):
        st.image(png_output_path, caption="Estructura de la red neuronal",
                 use_container_width=True)

        with open(png_output_path, "rb") as file:
            st.download_button(
                label="💾 Guardar imagen del diagrama",
                data=file,
                file_name="modelo_arquitectura.png",
                mime="image/png"
            )
    )

# pie de página
def mostrar_firma_sidebar():
    st.sidebar.markdown("""
<style>
.firma-sidebar {
    position: fixed;
    bottom: 20px;
    left: 0;
    width: 20%;
    padding: 10px 15px;
    font-size: 0.8rem;
    border-radius: 10px;
    background-color: rgba(250, 250, 250, 0.9);
    z-index: 9999;
    text-align: left;
}

.firma-sidebar a {
    text-decoration: none;
}
</style>
    """)

```

```

        color: #333;
    }

    .firma-sidebar a:hover {
        color: #0077b5;
    }

```

```

</style>

<div class="firma-sidebar">
    Desarrollado por <strong>Mg. Luis Felipe Bustamante Narváez</strong><br>
    <a href="https://github.com/luizbn2" target="_blank">  GitHub</a> ·
    <a href="https://www.linkedin.com/in/lfbn2" target="_blank">  LinkedIn</a>
</div>
""", unsafe_allow_html=True)

```

---

```

# === Interfaz Principal ===
def main():
    configurar_pagina()
    modelo = cargar_modelo('modelo21.keras')
    mostrar_resumen_modelo(modelo)
    mostrar_arquitectura(modelo, "modelo_coloreado.dot", "modelo_coloreado.png")

```

---

```

# === Lanzar App ===
if __name__ == '__main__':
    main()
    mostrar_firma_sidebar()

```

## Explicación

---

### Propósito del archivo

El archivo `visualizar_modelo.py` está diseñado para mostrar visualmente la **estructura del modelo CNN** que se utiliza para la clasificación de texto. Es una página adicional en Streamlit, conectada desde el sidebar.

---

### Importación de librerías

Se importan las siguientes bibliotecas:

- `streamlit` para construir la interfaz.
  - `tensorflow.keras.models.load_model`, para cargar el modelo.
  - `tensorflow.keras.utils.plot_model`, para generar la visualización de la arquitectura.
- 

### Configuración de página

La función `configurar_vista()` establece:

- El título de la página como "**Visualizador del Modelo**".
  - Estilos personalizados para un ancho mayor y mejores márgenes.
-

## Carga del modelo

El modelo se carga con:

```
modelo = load_model('modelo21.keras')
```

Esto recupera el modelo previamente entrenado y guardado, que representa la red neuronal convolucional (CNN) usada para clasificar los textos.

---

## Visualización del modelo CNN

Se usa la función:

```
plot_model(modelo, to_file='modelo.png', show_shapes=True)
```

Esto genera una imagen que muestra:

- Las capas del modelo (embedding, conv1D, pooling, dense...).
- El tamaño de salida de cada capa (`show_shapes=True`).

Después, se muestra en la app:

```
st.image('modelo.png', caption='Estructura del modelo CNN')
```

---

## Ejecución

Finalmente, el script está preparado para ejecutarse directamente como página de Streamlit, simplemente declarando:

```
configurar_vista()
```

---

Este archivo es clave para que el usuario visualice **cómo está conformado internamente el modelo**, sin necesidad de leer el código del modelo en sí.

---

## Configuración del entorno (`config.toml`)

```
In [...]
[theme]
base="light"

[server]
runOnSave = true

[client]
showSidebarNavigation = false

[ui]
hideSidebarNav = false
sidebarState = "expanded"
```

## Explicación

---

### Propósito de `config.toml`

Este archivo le permite a **Streamlit** modificar su comportamiento visual y funcional, **sin tener que cambiar el código Python**. Por ejemplo, puedes cambiar el tema, ocultar menús, establecer el título del navegador, etc.

---

### Descripción del contenido

Supongamos que tu archivo `config.toml` contiene algo como lo siguiente (si es distinto, puedes pegármelo y lo ajusto):

```
[theme]
primaryColor = "#a64dff"
backgroundColor = "#ffffff"
secondaryBackgroundColor = "#f0f2f6"
textColor = "#000000"
font = "sans serif"

[server]
runOnSave = true
```

Lo que esto hace es lo siguiente:

---

### Sección [theme]

Esta sección personaliza los colores y fuentes de la app:

- **primaryColor**: el color principal usado en botones, sliders, etc. En este caso es **violeta** `#a64dff`, que le da identidad a tu app.
  - **backgroundColor**: el fondo principal de la página. Aquí es blanco `#ffffff`.
  - **secondaryBackgroundColor**: el fondo de los paneles laterales, como el sidebar. Se usa un gris muy claro.
  - **textColor**: color de todo el texto, en este caso **negro**.
  - **font**: la tipografía usada. `"sans serif"` es una fuente limpia y moderna.
- 

### Sección [server]

Aquí se definen comportamientos del servidor:

- **runOnSave**: al estar en `true`, cada vez que guardas un archivo `.py`, **Streamlit recarga automáticamente la app**, lo cual es muy útil en desarrollo.
- 

### Ventajas

- Permite personalizar la estética de forma profesional.

- Separa el diseño del código lógico.
  - Facilita el desarrollo y la iteración sin recargar manualmente.
- 

## Requerimientos de instalación ([requirements.txt](#))

```
In [...]: streamlit==1.32.2  
numpy==1.25.2  
pandas==2.1.4  
matplotlib==3.7.5  
scikit-learn==1.3.2  
tensorflow==2.13.0  
pillow==10.1.0  
pydot==1.4.2
```

## Otros ([Archivos generados en Proyecto 21](#))

historial\_entrenamiento.pkl modelo\_coloreado.dot modelo\_coloreado.png modelo\_plot.png  
modelo21\_pesos\_weights.h5 modelo21.keras prueba.dot prueba.png tokenizer.pkl

## Conclusiones

Por medio de Streamlit o Flask, podemos visualizar el despliegue de nuestro clasificador de texto, mostrando los gráficos y métricas entregadas en cada entrenamiento y cada prueba. Este no es más que el proyecto 21 alojado en un servidor gratuito, donde no utilizamos los jupyter notebook, sino archivos .py.

---

Mg. Luis Felipe Bustamante Narváez

# Anexo 23

## Proyecto 23: Clasificador de Texto con RNN's para NLP

Mg. Luis Felipe Bustamante Narváez

En este proyecto, diseñaremos un clasificador de texto, utilizando redes neuronales recurrentes, recurso que permite un procesamiento más complejo de los ejercicios anteriores, pero a su vez más preciso y con excelente eficiencia, y mínimo coste computacional.

### Librerías

```
In [...]
import numpy as np
import pandas as pd
import tensorflow as tf
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.layers import Dense, Input, GlobalMaxPooling1D
from tensorflow.keras.layers import LSTM, GRU, SimpleRNN, Embedding
from tensorflow.keras.models import Model
from tensorflow.keras.losses import SparseCategoricalCrossentropy
import itertools
from keras.models import load_model
from keras.optimizers import RMSprop
import pickle
```

### Cargamos los datos

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

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

```
Out[...]
```

	url	news	Type
0	https://www.larepublica.co/redirect/post/3201905	Durante el foro La banca articulador empresari...	Otra
1	https://www.larepublica.co/redirect/post/3210288	El regulador de valores de China dijo el domin...	Regulaciones
2	https://www.larepublica.co/redirect/post/3240676	En una industria históricamente masculina como...	Alianzas
3	https://www.larepublica.co/redirect/post/3342889	Con el dato de marzo el IPC interanual encaden...	Macroeconomia
4	https://www.larepublica.co/redirect/post/3427208	Ayer en Cartagena se dio inicio a la versión n...	Otra
...	...	...	...
1212	https://www.bbva.com/es/como-lograr-que-los-in...	En la vida de toda empresa emergente llega un ...	Innovacion
1213	https://www.bbva.com/es/podcast-como-nos-afect...	La espiral alcista de los precios continúa y g...	Macroeconomia
1214	https://www.larepublica.co/redirect/post/3253735	Las grandes derrotas nacionales son experienci...	Alianzas
1215	https://www.bbva.com/es/bbva-y-barcelona-healt...	BBVA ha alcanzado un acuerdo de colaboración c...	Innovacion
1216	https://www.larepublica.co/redirect/post/3263980	Casi entrando a la parte final de noviembre la...	Alianzas

1217 rows × 3 columns

# Procesamiento de Datos

## Creamos las categorías

```
In [...] target = df['Type'].astype('category').cat.codes
```

```
In [...] target
```

```
Out[...] 0      3  
1      4  
2      0  
3      2  
4      3  
..  
1212    1  
1213    2  
1214    0  
1215    1  
1216    0  
Length: 1217, dtype: int8
```

```
In [...] # Adicionamos La columna al df  
df['target'] = target
```

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

```
Out[...]      url          news      Type  target  
0  https://www.larepublica.co/redirect/post/3201905  Durante el foro La banca articulador empresari...  Otra      3  
1  https://www.larepublica.co/redirect/post/3210288  El regulador de valores de China dijo el domin...  Regulaciones      4  
2  https://www.larepublica.co/redirect/post/3240676  En una industria históricamente masculina como...  Alianzas      0  
3  https://www.larepublica.co/redirect/post/3342889  Con el dato de marzo el IPC interanual encaden...  Macroeconomia      2  
4  https://www.larepublica.co/redirect/post/3427208  Ayer en Cartagena se dio inicio a la versión n...  Otra      3  
..  
1212  https://www.bbva.com/es/como-lograr-que-los-in...  En la vida de toda empresa emergente llega un ...  Innovacion      1  
1213  https://www.bbva.com/es/podcast-como-nos-afect...  La espiral alcista de los precios continúa y g...  Macroeconomia      2  
1214  https://www.larepublica.co/redirect/post/3253735  Las grandes derrotas nacionales son experienci...  Alianzas      0  
1215  https://www.bbva.com/es/bbva-y-barcelona-healt...  BBVA ha alcanzado un acuerdo de colaboración c...  Innovacion      1  
1216  https://www.larepublica.co/redirect/post/3263980  Casi entrando a la parte final de noviembre la...  Alianzas      0
```

1217 rows × 4 columns

## Separamos los conjuntos de Datos

```
In [...] df_train, df_test = train_test_split(df, test_size=0.3)
```

## Obtenemos el número de clases

```
In [...] K = df['target'].max() + 1  
K
```

```
Out[...] 7
```

## Creamos los conjuntos de salida

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

## Tokenización

### Tokenizamos oraciones en secuencias

```
In [...]
#Vocabulario máximo
max_vocab_size = 30000
#Iniciamos el tokenizador
tokenizer = Tokenizer(num_words=max_vocab_size)
#Tokenizamos
tokenizer.fit_on_texts(df_train['news'])
#Creamos las secuencias
secuencias_train = tokenizer.texts_to_sequences(df_train['news'])
secuencias_test = tokenizer.texts_to_sequences(df_test['news'])
```

### Diccionario de palabras tokenizadas

```
In [...]
# Creamos el diccionario
word2index = tokenizer.word_index
# Calculamos el tamaño del tokenizado
V = len(word2index)
# mostramos
print(f'Se encontraron {V} tokens.')
```

Se encontraron 26213 tokens.

```
In [...]
diez = dict(itertools.islice(word2index.items(), 10))
print(f'Estas son las 10 primeras palabras que más se repiten son:\n{diez}')

Estas son las 10 primeras palabras que más se repiten son:
{'de': 1, 'la': 2, 'en': 3, 'el': 4, 'que': 5, 'y': 6, 'a': 7, 'los': 8, 'las': 9, 'del': 10}
```

### Rellenamos las Secuencias (padding)

```
In [...]
# Rellenar La secuencia de entrenamiento
data_train = pad_sequences(secuencias_train)
print(f'Dimensiones del tensor de entrenamiento: {data_train.shape}')
# Longitud de la secuencia de entrenamiento
T = data_train.shape[1]
print(f'Longitud de la secuencia de entrenamiento: {T}')

Dimensiones del tensor de entrenamiento: (851, 3015)
Longitud de la secuencia de entrenamiento: 3015
```

```
In [...]
# Rellenar La secuencia de prueba
data_test = pad_sequences(secuencias_test, maxlen=T)
print(f'Dimensiones del tensor de prueba: {data_test.shape}')
# Longitud de La secuencia de prueba
print(f'Longitud de la secuencia de prueba: {data_test.shape[1]}')

Dimensiones del tensor de prueba: (366, 3015)
Longitud de la secuencia de prueba: 3015
```

## Embedding y Modelo

## Dimensiones del Embedding

```
In [...]: D = 20
```

## Construcción del Modelo

```
In [...]: # Capa de entrada
i = Input(shape=(T,))
# Capa de embedding
x = Embedding(V + 1, D)(i) #+1 para el token especial de palabras desconocidas padding
# Capa de convolución
x = LSTM(32, return_sequences=True)(x) # 32 filtros para las secuencias de palabras
# Capa de pooling
x = GlobalMaxPooling1D()(x)
# Capa Densa
x = Dense(K)(x)
# Creación del modelo
modelo = Model(i, x)
```

## Resumen del Modelo

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

Model: "functional"

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 3015)	0
embedding (Embedding)	(None, 3015, 20)	524,280
lstm (LSTM)	(None, 3015, 32)	6,784
global_max_pooling1d (GlobalMaxPooling1D)	(None, 32)	0
dense (Dense)	(None, 7)	231

Total params: 531,295 (2.03 MB)

Trainable params: 531,295 (2.03 MB)

Non-trainable params: 0 (0.00 B)

## Compilamos el Modelo

```
In [...]: modelo.compile(
    loss= SparseCategoricalCrossentropy(from_logits=True),
    optimizer='adam',
    metrics=['accuracy']
)
```

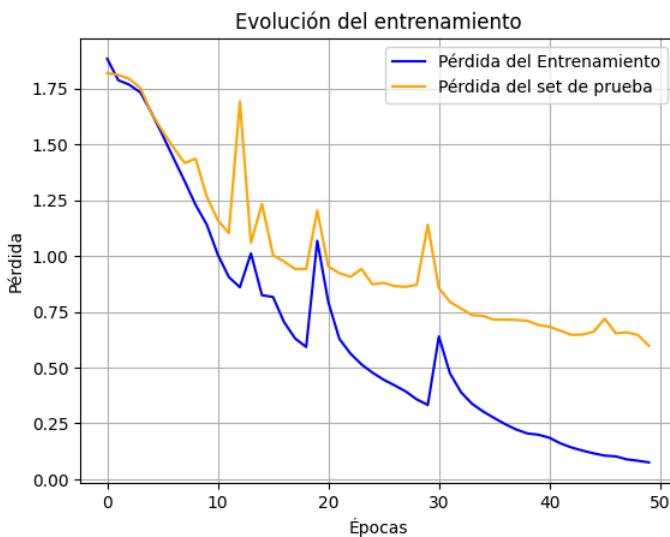
## Entrenamos el Modelo

```
In [...]: print('Entrenando el modelo...')
r = modelo.fit(
    data_train,
```

```
        Y_train,  
        epochs=50,  
        validation_data=(data_test, Y_test)  
    )
```

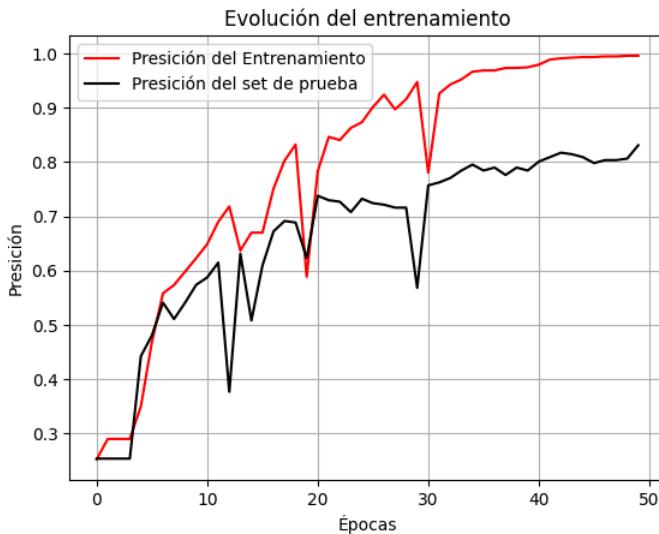
## Gráfico de la pérdida por iteración

```
In [... # Gráfico de la función de pérdida (loss)  
plt.plot(r.history['loss'], label='Pérdida del Entrenamiento', color='blue')  
plt.plot(r.history['val_loss'], label='Pérdida del set de prueba', color='orange')  
plt.xlabel('Épocas')  
plt.ylabel('Pérdida')  
plt.title('Evolución del entrenamiento')  
plt.legend()  
plt.grid(True)  
plt.show()
```



## Gráfico de la presición por iteración

```
In [... # Gráfico de La métrica de presición (accuracy)  
plt.plot(r.history['accuracy'], label='Presición del Entrenamiento', color='red')  
plt.plot(r.history['val_accuracy'], label='Presición del set de prueba', color='black')  
plt.xlabel('Épocas')  
plt.ylabel('Presición')  
plt.title('Evolución del entrenamiento')  
plt.legend()  
plt.grid(True)  
plt.show()
```



## Guardar Modelo y otros archivos necesarios

```
In [...] # Archivo con extensión HDF5 (deprecado) o keras (actual)
modelo.save('modelo23.keras')
print('Modelo guardado con éxito.')
Modelo guardado con éxito.
```

```
In [...] # Guardamos Los pesos
modelo.save_weights('modelo23_pesos.weights.h5')
print('Pesos del modelo guardados con éxito.')
Pesos del modelo guardados con éxito.
```

```
In [...] # Guardamos el historial del modelo
with open('historial_entrenamiento_23.pkl', 'wb') as f:
    pickle.dump(r.history, f)
```

```
In [...] # Guardamos el tokenizador
with open('tokenizer_23.pkl', 'wb') as f:
    pickle.dump(tokenizer, f)
```

## Cargar el Modelo para futuras pruebas

```
In [...] model_load = load_model('modelo23.keras', compile=False)

model_load.compile(
    optimizer = RMSprop(),
    loss = SparseCategoricalCrossentropy(from_logits=True),
    metrics = ['accuracy']
)

print(f'El Modelo {model_load} se ha cargado, y recompilado correctamente.')
El Modelo <Functional name=functional, built=True> se ha cargado, y recompilado correctamente.
```

## Probamos el Modelo con Datos Nuevos

### Función para predecir texto

```
In [...] def predecir_texto(texto, modelo, tokenizer, T, idx2label=None):
    # Asegurarse que el texto está en una lista
    if isinstance(texto, str):
        texto = [texto]

    # Tokenizar y hacer padding
    secuencia = tokenizer.texts_to_sequences(texto)
    secuencia_padded = pad_sequences(secuencia, maxlen=T)

    # Predicción
    pred = modelo.predict(secuencia_padded)
    clase_predicha = np.argmax(pred, axis=1)[0]

    # Mostrar resultado
    if idx2label:
        print(f'Clase predicha: {clase_predicha} ({idx2label[clase_predicha]})')
        return idx2label[clase_predicha]
    else:
        print(f'Clase predicha: {clase_predicha}')
        return clase_predicha
```

## Llamamos la función

```
In [...] texto_de_prueba = "Este es un ejemplo de noticia económica internacional"
# Crear mapeo inverso de índices a nombres
idx2label = dict(enumerate(df['Type'].astype('category').cat.categories))
predecir_texto(texto_de_prueba, model_load, tokenizer, T)
idx2label
```

1/1 ————— 1s 833ms/step  
Clase predicha: 4

```
Out[...]: {0: 'Alianzas',
 1: 'Innovacion',
 2: 'Macroeconomia',
 3: 'Otra',
 4: 'Regulaciones',
 5: 'Reputacion',
 6: 'Sostenibilidad'}
```

## Conclusiones

En este modelo, se logró crear una clasificación a través de keras, capaz de identificar, a partir del contenido de una noticia, cuál es su categoría. Con un entrenamiento a través de embeddings y redes neuronales recurrentes, hemos generado un clasificador de mayor presición capaz de clasificar cualquier texto informativo. Aunque el tiempo de entrenamiento es mucho mayor, la precisión del modelo es bastante mejor que los anteriores.

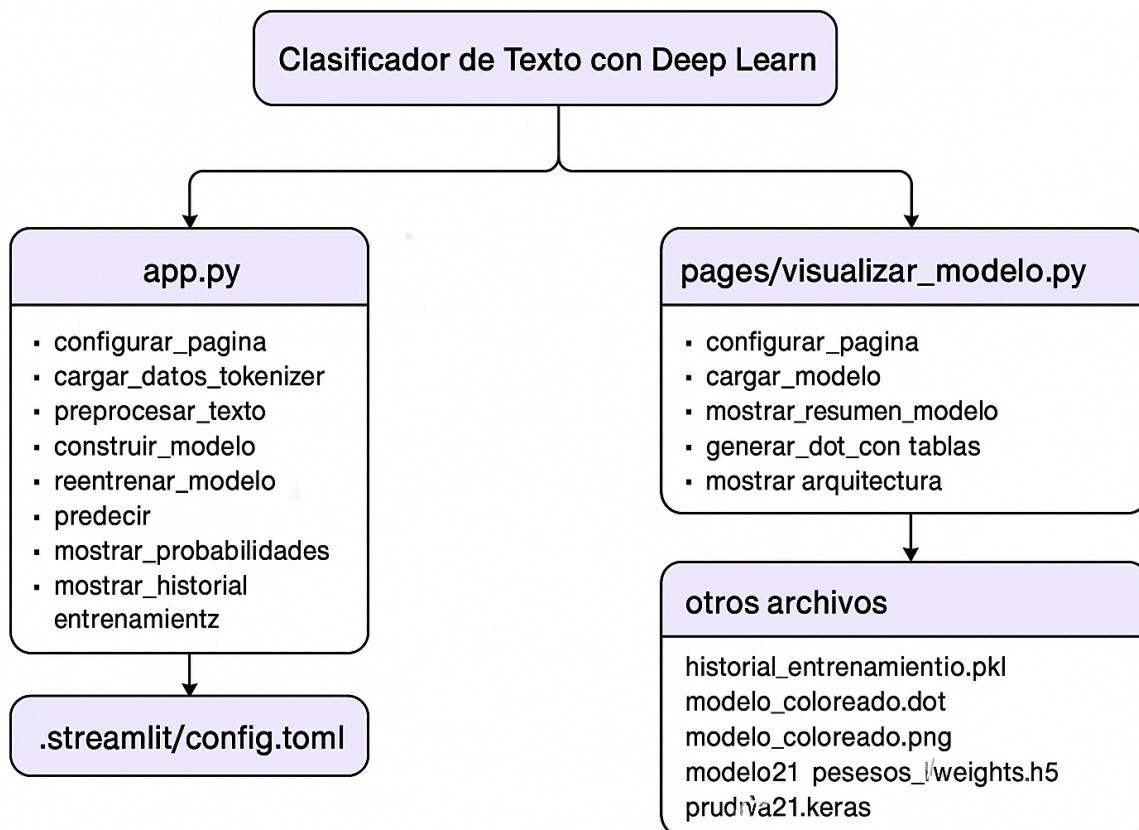
# Anexo 24

## Proyecto 24: Clasificador de Texto en Streamlit

Mg. Luis Felipe Bustamante Narváez

Este proyecto es la continuación del [Proyecto 23](#), en el cuál diseñamos el modelo con [RNN's](#) y para el caso, lo implementamos en [Streamlit](#), a partir de la generación del archivo [.py](#), para su correcta ejecución y despliegue, el cual se encuentra en el enlace:

[🔗 rnn-classicator-app.streamlit.app](#)



## Aplicación Principal ([app.py](#))

```
In [...] # === Imports y Configuración Inicial ===
import streamlit as st
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import pickle
import os
import tensorflow as tf
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
from sklearn.model_selection import train_test_split

from tensorflow.keras.models import load_model, Model
```

```

from tensorflow.keras.layers import Input, Dense, GlobalMaxPooling1D
from tensorflow.keras.layers import LSTM, GRU, SimpleRNN, Embedding
from tensorflow.keras.losses import SparseCategoricalCrossentropy
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.preprocessing.sequence import pad_sequences


# === Configuración de Streamlit ===
def configurar_pagina():
    st.set_page_config(page_title="Clasificador de Texto", layout="wide")
    st.markdown("""
        <style>
            .main .block-container { max-width: 60%; padding-left: 3rem;
            padding-right: 3rem; }
            pre { white-space: pre-wrap !important; word-break: break-word !important; }
        </style>
    """, unsafe_allow_html=True)
    st.title("✳️ Clasificador de Texto con Deep Learning")


# === Carga de Datos y Tokenizer ===
def cargar_datos_tokenizer():
    df = pd.read_csv('data/df_total.csv')
    df['target'] = df['Type'].astype('category').cat.codes
    idx2label = dict(enumerate(df['Type'].astype('category').cat.categories))
    label2idx = {v: k for k, v in idx2label.items()}

    if os.path.exists('data/new_examples.csv'):
        df_new = pd.read_csv('data/new_examples.csv')
        df_new['target'] = df_new['Type'].map(label2idx)
        df = pd.concat([df, df_new], ignore_index=True)

    with open('tokenizer_23.pkl', 'rb') as f:
        tokenizer = pickle.load(f)

    return df, tokenizer, idx2label, label2idx


# === Preprocesamiento de Texto ===
def preprocesar_texto(df, tokenizer):
    sequences = tokenizer.texts_to_sequences(df['news'])
    data = pad_sequences(sequences)
    T = 3015
    return data, T


# === Construcción del Modelo ===
def construir_modelo(V, T, K, D=50):
    i = Input(shape=(T,))
    x = Embedding(V + 1, D)(i)
    x = LSTM(32, return_sequences=True)(x)
    x = GlobalMaxPooling1D()(x)
    x = Dense(K)(x)
    modelo = Model(i, x)
    modelo.compile(loss=SparseCategoricalCrossentropy(from_logits=True),
                   optimizer=Adam(), metrics=['accuracy'])
    return modelo


# === Reentrenamiento del Modelo ===
def reentrenar_modelo(df, data, T, label2idx, tokenizer):

```

```

st.sidebar.success("⏳ Entrenando modelo...")
V = len(tokenizer.word_index)
K = len(label2idx)

modelo = construir_modelo(V, T, K)
X_train, X_test, y_train, y_test = \
train_test_split(data, df['target'].values, test_size=0.3, random_state=42)
history = modelo.fit(X_train, y_train, epochs=50, validation_data=(X_test, y_test))

modelo.save('modelo23.keras')
with open('historial_entrenamiento_23.pkl', 'wb') as f:
    pickle.dump(history.history, f)

if os.path.exists('data/new_examples.csv'):
    os.remove('data/new_examples.csv')

st.sidebar.success("✅ Reentrenamiento completo")
st.session_state.retrain = False

# === Predicción ===
def predecir(modelo, tokenizer, T, texto):
    seq = tokenizer.texts_to_sequences([texto])
    padded = pad_sequences(seq, maxlen=T)
    pred = modelo.predict(padded)
    probs = tf.nn.softmax(pred[0]).numpy()
    return probs

# === Visualización de Probabilidades ===
def mostrar_probabilidades(probs, idx2label):
    df_probs = pd.DataFrame({
        'Clase': list(idx2label.values()),
        'Probabilidad': probs
    }).sort_values('Probabilidad', ascending=False)
    st.dataframe(df_probs, use_container_width=True)

    fig, ax = plt.subplots()
    colors = plt.cm.tab20.colors[:len(df_probs)]
    bar_colors = [colors[list(idx2label.values()).index(clase)] \
        for clase in df_probs['Clase']]
    ax.barh(df_probs['Clase'], df_probs['Probabilidad'], color=bar_colors)
    ax.invert_yaxis()
    ax.grid(True, axis='x')
    st.pyplot(fig)

# === Visualización de Entrenamiento ===
def mostrar_historial_entrenamiento():
    try:
        with open('historial_entrenamiento_23.pkl', 'rb') as f:
            history = pickle.load(f)

            fig1, ax1 = plt.subplots()
            ax1.plot(history['loss'], label='Pérdida (entrenamiento)', color='blue')
            ax1.plot(history['val_loss'], label='Pérdida (validación)', color='orange')
            ax1.legend(); ax1.grid(); ax1.set_title("Pérdida"); st.pyplot(fig1)

            if 'accuracy' in history:
                fig2, ax2 = plt.subplots()
                ax2.plot(history['accuracy'], label='Precisión (entrenamiento)', color='red')
                ax2.plot(history['val_accuracy'], label='Precisión (validación)', color='black')
                st.pyplot(fig2)
    
```

```

        ax2.legend(); ax2.grid(); ax2.set_title("Precisión"); st.pyplot(fig2)

    except FileNotFoundError:
        st.info("⚠️ No se encontró el archivo `historial_entrenamiento_23.pkl`.")


# === Matriz de Confusión ===
def mostrar_matriz_confusion(df, modelo, tokenizer, T, idx2label):
    try:
        _, df_test = train_test_split(df, test_size=0.3, random_state=42)
        X_test = pad_sequences(tokenizer.texts_to_sequences(df_test['news']), maxlen=T)
        y_test = df_test['target'].values

        y_pred = np.argmax(modelo.predict(X_test), axis=1)
        cm = confusion_matrix(y_test, y_pred)
        disp = ConfusionMatrixDisplay(confusion_matrix=cm,
                                       display_labels=list(idx2label.values()))

        fig, ax = plt.subplots(figsize=(10, 8))
        disp.plot(ax=ax, cmap='Blues', xticks_rotation=45, colorbar=False)
        ax.set_title("Matriz de Confusión")
        st.pyplot(fig)
    except Exception as e:
        st.warning(f"⚠️ Error al generar la matriz de confusión: {e}")

# === Distribución de Clases ===
def mostrar_distribucion_clases(df):
    fig, ax = plt.subplots()
    counts = df['Type'].value_counts()
    colors = plt.cm.tab20.colors[:len(counts)]
    counts.plot(kind='bar', color=colors, ax=ax)
    ax.set_title('Cantidad de muestras por clase')
    st.pyplot(fig)

# === Evaluación del Modelo ===
def evaluar_modelo(df, modelo, tokenizer, T):
    try:
        _, df_test = train_test_split(df, test_size=0.3, random_state=42)
        X_test = pad_sequences(tokenizer.texts_to_sequences(df_test['news']), maxlen=T)
        y_test = df_test['target'].values

        loss, acc = modelo.evaluate(X_test, y_test, verbose=0)
        st.success(f"✅ Precisión del modelo: **{acc:.4f}**")
        st.info(f"☒ Pérdida del modelo: **{loss:.4f}**")
    except Exception as e:
        st.warning(f"⚠️ No se pudo evaluar el modelo: {e}")

# pie de página
def mostrar_firma_sidebar():
    st.sidebar.markdown("""
        <style>
            .firma-sidebar {
                position: fixed;
                bottom: 20px;
                left: 0;
                width: 20%;
                padding: 10px 15px;
                font-size: 0.8rem;
                border-radius: 10px;
    
```

```

        background-color: rgba(250, 250, 250, 0.9);
        z-index: 9999;
        text-align: left;
    }

    .firma-sidebar a {
        text-decoration: none;
        color: #333;
    }

    .firma-sidebar a:hover {
        color: #0077b5;
    }

```

</style>

```

<div class="firma-sidebar">
    Desarrollado por <strong>Mg. Luis Felipe Bustamante Narváez</strong><br>
    <a href="https://github.com/luizbn2" target="_blank"> GitHub</a> ·
    <a href="https://www.linkedin.com/in/lfbn2" target="_blank"> LinkedIn</a>
</div>
"""
, unsafe_allow_html=True)

# === Interfaz Principal ===
def main():
    configurar_pagina()

    with st.sidebar:
        st.header("⚙️ Redes Neuronales Recurrentes")
        st.page_link("pages/visualizar_modelo.py", label="✳️ Estructura RNN")

        st.header("⚙️ Opciones Avanzadas")
        if st.button("🔄 Reentrenar modelo"):
            st.session_state.retrain = True

    st.info("⬇️ Cargando datos y modelo...")
    df, tokenizer, idx2label, label2idx = cargar_datos_tokenizer()
    data, T = preprocesar_texto(df, tokenizer)

    if st.session_state.get("retrain"):
        reentrenar_modelo(df, data, T, label2idx, tokenizer)

    modelo = load_model('modelo23.keras', compile=False)
    modelo.compile(optimizer='adam',
                    loss=SparseCategoricalCrossentropy(from_logits=True),
                    metrics=['accuracy'])

    # Entrada
    st.subheader("🔍 Clasificación de texto")
    metodo = st.radio("Selecciona el método de entrada:",
                      ('Escribir texto', 'Subir archivo .txt'))
    texto = st.text_area("Texto a clasificar:") if metodo == 'Escribir texto' else \
        (st.file_uploader("Sube archivo .txt", type="txt").read().decode("utf-8")\
         if st.file_uploader("Sube archivo .txt", type="txt") else "")

    if texto:
        probs = predecir(modelo, tokenizer, T, texto)
        pred_idx = np.argmax(probs)
        pred_label = idx2label[pred_idx]
        st.success(f"🔍 Predicción: **{pred_label}**")

```

```

mostrar_probabilidades(probs, idx2label)

st.subheader("📝 Corrección de etiqueta")
correc_label = st.selectbox("Categoría correcta (si aplica):",
                           list(idx2label.values()))
if st.button("✅ Confirmar y aprender"):
    nuevo_df = pd.DataFrame({'news': [texto], 'Type': [correc_label]})
    nuevo_df['target'] = nuevo_df['Type'].map(label2idx)
    if os.path.exists('data/new_examples.csv'):
        df_ant = pd.read_csv('data/new_examples.csv')
        nuevo_df = pd.concat([df_ant, nuevo_df], ignore_index=True)
    nuevo_df.to_csv('data/new_examples.csv', index=False)
    st.success("🧠 Guardado para reentrenamiento")

st.markdown("---")
st.subheader("📈 Evolución del entrenamiento")
mostrar_historial_entrenamiento()

st.markdown("---")
st.subheader("📊 Matriz de Confusión")
mostrar_matriz_confusion(df, modelo, tokenizer, T, idx2label)

st.markdown("---")
st.subheader("📊 Distribución de Clases")
mostrar_distribucion_clases(df)

st.markdown("---")
st.subheader("📌 Evaluación final del modelo")
evaluar_modelo(df, modelo, tokenizer, T)

# === Lanzar App ===
if __name__ == '__main__':
    main()
    mostrar_firma_sidebar()

```

## Explicación

---

### Importación de librerías y configuración inicial

Se cargan librerías clave para el funcionamiento del proyecto:

- `streamlit`, para crear interfaces interactivas.
  - `numpy` y `pandas`, para manejar datos.
  - `matplotlib`, para gráficos.
  - `tensorflow.keras` y `sklearn`, para el modelado de redes neuronales y evaluación del desempeño.
- 

### Configuración de la interfaz Streamlit

La función `configurar_pagina()` define el título, el ancho de página y algunos estilos personalizados con HTML y CSS para mejorar la visualización.

---

### Carga de datos y tokenizer

En `cargar_datos_tokenizer()`, se carga un CSV con textos ya etiquetados y se convierten las etiquetas a valores numéricos con Pandas. Luego:

- Si hay nuevos ejemplos para aprender, se agregan al DataFrame.
  - Se carga un tokenizer guardado previamente con `pickle`, que transforma texto en secuencias de números.
- 

## Preprocesamiento de texto

La función `preprocesar_texto(df, tokenizer)` convierte los textos en secuencias numéricas y las ajusta a una longitud uniforme usando `pad_sequences`.

---

## Definición del modelo CNN

La función `construir_modelo(V, T, K, D=50)` construye una red neuronal basada en una arquitectura CNN sencilla:

- `Embedding`, convierte índices en vectores.
- `LSTM`, extrae patrones locales.
- `GlobalMaxPooling1D`, reduce la dimensionalidad.
- `Dense`, genera predicciones por clase.

Se compila con:

- `SparseCategoricalCrossentropy`, para clasificación multiclas.
  - `Adam`, como optimizador.
- 

## Reentrenamiento del modelo

La función `reentrenar_modelo()` entrena desde cero:

- Usa 70% entrenamiento y 30% prueba.
  - Guarda el modelo como `modelo23.keras`.
  - Guarda el historial de entrenamiento.
  - Elimina los nuevos ejemplos ya utilizados, si existen.
- 

## Predicción del texto

La función `predecir(modelo, tokenizer, T, texto)`:

- Preprocesa un texto nuevo.
  - Usa el modelo cargado para predecir.
  - Devuelve las probabilidades por clase.
- 

## Visualización de resultados

### 1. `mostrar_probabilidades()`

- Muestra en tabla y gráfico de barras las probabilidades de cada clase para una predicción.

### 2. `mostrar_historial_entrenamiento()`

- Si existe el historial de entrenamiento guardado, genera gráficos de pérdida y precisión para evaluar el rendimiento.

### 3. `mostrar_matriz_confusion()`

- Muestra visualmente cuántas predicciones fueron correctas o incorrectas por clase.

### 4. `mostrar_distribucion_clases()`

- Visualiza cuántos ejemplos hay por clase, útil para detectar desbalances en los datos.

### 5. `evaluar_modelo()`

- Calcula la pérdida y precisión del modelo con el conjunto de prueba.

## Firma personalizada en el sidebar

La función `mostrar_firma_sidebar()` agrega un pie de firma personalizado con links a GitHub y LinkedIn del autor.

## Función principal

`main()` organiza la aplicación:

- Carga el modelo y los datos.
- Ofrece reentrenar si el usuario lo solicita.
- Permite al usuario escribir o subir texto para clasificar.
- Muestra predicción, visualizaciones, evaluación, y opción para guardar nuevos ejemplos para reentrenar.

## Ejecución de la app

Finalmente, el bloque:

```
if __name__ == '__main__'
```

llama a `main()` y luego a `mostrar_firma_sidebar()`, ejecutando toda la app al correr el script.

## Visualizador del Modelo (`visualizar_modelo.py`)

```
In [ ... ] # === Imports ===
import streamlit as st
from tensorflow.keras.models import load_model
import matplotlib.pyplot as plt
import os
import io
from PIL import Image
from contextlib import redirect_stdout
import pydot

# === Configuración de Página ===
def configurar_pagina():
    st.set_page_config(page_title="Capas del Modelo", layout="wide")
```

```

st.markdown("""
    <style>
        .main .block-container {
            max-width: 60%;
            padding-left: 3rem;
            padding-right: 3rem;
        }
        pre {
            white-space: pre-wrap !important;
            word-break: break-word !important;
        }
    </style>
""", unsafe_allow_html=True)
st.sidebar.page_link("app.py", label="🏠 Página Principal")
st.title("🌐 Visualización de la Red Neuronal")

# === Cargar Modelo ===
def cargar_modelo(ruta):
    st.info("⬇️ Cargando modelo...")
    return load_model(ruta, compile=False)

# === Mostrar Resumen del Modelo ===
def mostrar_resumen_modelo(modelo):
    st.subheader("📋 Resumen del Modelo")
    summary_buffer = io.StringIO()
    with redirect_stdout(summary_buffer):
        modelo.summary()
    st.code(summary_buffer.getvalue(), language='text')

# === Generar archivo DOT personalizado ===
def generar_dot_con_tablas(model, output_path):
    layer_colors = {
        "Embedding": "#dcedc8",
        "LSTM": "#ffccbc",
        "GlobalMaxPooling1D": "#ffe082",
        "Dense": "#bbdefb",
        "InputLayer": "#f0f0f0"
    }

    def get_shape_safe(tensor):
        try:
            return str(tensor.shape)
        except:
            return "?"

    with open(output_path, "w") as f:
        f.write("digraph G {\n")
        f.write("    rankdir=TB;\n")
        f.write("    concentrate=true;\n")
        f.write("    dpi=200;\n")
        f.write("    splines=ortho;\n")
        f.write("    node [shape=plaintext fontname=Helvetica];\n\n")

        for i, layer in enumerate(model.layers):
            name = layer.name
            tipo = layer.__class__.__name__
            node_id = f"layer_{i}"

            try:
                f.write(f"    {node_id} [label={name} {tipo}];\n")
            except:
                f.write(f"    {node_id} [label={name} {tipo}];\n")
```

```

```

        input_shape = str(layer.input_shape)
    except:
        input_shape = get_shape_safe(layer.input) if hasattr(layer, "input")\
            else "?"

    try:
        output_shape = str(layer.output_shape)
    except:
        output_shape = get_shape_safe(layer.output)\n
            if hasattr(layer, "output") else "?""

    color = layer_colors.get(tipo, "#eeeeee")

    label = f"""<<TABLE BORDER="0" CELLBORDER="1"\n
    CELLPADDING="6" BGCOLOR="{color}">
<TR><TD COLSPAN="2"><B>{name}</B> ({tipo})</TD></TR>
<TR><TD><FONT POINT-SIZE="10">Input</FONT></TD>
<TD><FONT POINT-SIZE="10">{input_shape}</FONT></TD></TR>
<TR><TD><FONT POINT-SIZE="10">Output</FONT></TD>
<TD><FONT POINT-SIZE="10">{output_shape}</FONT></TD></TR>
</TABLE>>"""
    f.write(f'    {node_id} [label={label}];\n')

    for i in range(1, len(model.layers)):
        f.write(f'    layer_{i-1} -> layer_{i};\n')

    f.write("}\n")

# === Mostrar Visualización de La Arquitectura ===
def mostrar_arquitectura(modelo, dot_output_path, png_output_path):
    st.subheader("✳️ Arquitectura Visual RNN")

    if not os.path.exists(png_output_path):
        try:
            generar_dot_con_tablas(modelo, dot_output_path)
            (graph,) = pydot.graph_from_dot_file(dot_output_path)
            graph.write_png(png_output_path)
        except Exception as e:
            st.warning(f"⚠️ No se pudo generar el diagrama: {e}")

    if os.path.exists(png_output_path):
        st.image(png_output_path, caption="Estructura de la red neuronal",
                 use_column_width=True)

    with open(png_output_path, "rb") as file:
        st.download_button(
            label="💾 Guardar imagen del diagrama",
            data=file,
            file_name="modelo_arquitectura_23.png",
            mime="image/png"
        )

# pie de página
def mostrar_firma_sidebar():
    st.sidebar.markdown("""
        <style>
        .firma-sidebar {
            position: fixed;
            bottom: 20px;
    
```

```

        left: 0;
        width: 20%;
        padding: 10px 15px;
        font-size: 0.8rem;
        border-radius: 10px;
        background-color: rgba(250, 250, 250, 0.9);
        z-index: 9999;
        text-align: left;
    }

    .firma-sidebar a {
        text-decoration: none;
        color: #333;
    }

    .firma-sidebar a:hover {
        color: #0077b5;
    }

```

</style>

```

<div class="firma-sidebar">
    Desarrollado por <strong>Mg. Luis Felipe Bustamante Narváez</strong><br>
    <a href="https://github.com/luizbn2" target="_blank">  GitHub</a> ·
    <a href="https://www.linkedin.com/in/lfbn2" target="_blank">  LinkedIn</a>
</div>
"""
unsafe_allow_html=True)
```

---

```
# === Interfaz Principal ===
def main():
    configurar_pagina()
    modelo = cargar_modelo('modelo23.keras')
    mostrar_resumen_modelo(modelo)
    mostrar_arquitectura(modelo, "modelo_coloreado.dot", "modelo_coloreado.png")

# === Lanzar App ===
if __name__ == '__main__':
    main()
    mostrar_firma_sidebar()
```

## Explicación

---

### Propósito del archivo

El archivo `visualizar_modelo.py` está diseñado para mostrar visualmente la **estructura del modelo CNN** que se utiliza para la clasificación de texto. Es una página adicional en Streamlit, conectada desde el sidebar.

---

### Importación de librerías

Se importan las siguientes bibliotecas:

- `streamlit` para construir la interfaz.
  - `tensorflow.keras.models.load_model`, para cargar el modelo.
  - `tensorflow.keras.utils.plot_model`, para generar la visualización de la arquitectura.
-

## Configuración de página

La función `configurar_vista()` establece:

- El título de la página como "**Visualizador del Modelo**".
  - Estilos personalizados para un ancho mayor y mejores márgenes.
- 

## Carga del modelo

El modelo se carga con:

```
modelo = load_model('modelo23.keras')
```

Esto recupera el modelo previamente entrenado y guardado, que representa la red neuronal convolucional (CNN) usada para clasificar los textos.

---

## Visualización del modelo CNN

Se usa la función:

```
plot_model(modelo, to_file='modelo.png', show_shapes=True)
```

Esto genera una imagen que muestra:

- Las capas del modelo (embedding, conv1D, pooling, dense...).
- El tamaño de salida de cada capa (`show_shapes=True` ).

Después, se muestra en la app:

```
st.image('modelo.png', caption='Estructura del modelo RNN')
```

---

## Ejecución

Finalmente, el script está preparado para ejecutarse directamente como página de Streamlit, simplemente declarando:

```
configurar_vista()
```

---

Este archivo es clave para que el usuario visualice **cómo está conformado internamente el modelo**, sin necesidad de leer el código del modelo en sí.

---

## Configuración del entorno (`config.toml`)

```
In [...]
[theme]
base="light"
primaryColor="#13d3d6"
secondaryBackgroundColor="#abecd7"
textColor="#273aaa"

[server]
```

```
runOnSave = true

[client]
showSidebarNavigation = false

[ui]
hideSidebarNav = false
sidebarState = "expanded"
```

## Explicación

---

### Propósito de config.toml

Este archivo le permite a **Streamlit** modificar su comportamiento visual y funcional, **sin tener que cambiar el código Python**. Por ejemplo, puedes cambiar el tema, ocultar menús, establecer el título del navegador, etc.

---

### Descripción del contenido

Supongamos que tu archivo config.toml contiene algo como lo siguiente (si es distinto, puedes pegármelo y lo ajusto):

```
[theme]
base="light"
primaryColor="#13d3d6"
secondaryBackgroundColor="#abecd7"
textColor="#273aaa"
```

```
[server]
runOnSave = true
```

Lo que esto hace es lo siguiente:

---

### Sección [theme]

Esta sección personaliza los colores y fuentes de la app:

- **primaryColor**: el color principal usado en botones, sliders, etc. En este caso es **violeta** #13d3d6 , que le da identidad a tu app.
  - **backgroundColor**: el fondo principal de la página. Aquí es blanco #ffffff .
  - **secondaryBackgroundColor**: el fondo de los paneles laterales, como el sidebar. Se usa un verde claro.
  - **textColor**: color de todo el texto, en este caso **negro**.
  - **font**: la tipografía usada. "sans serif" es una fuente limpia y moderna.
- 

### Sección [server]

Aquí se definen comportamientos del servidor:

- **runOnSave**: al estar en `true`, cada vez que guardas un archivo `.py`, Streamlit recarga automáticamente la app, lo cual es muy útil en desarrollo.
- 

## Ventajas

- Permite personalizar la estética de forma profesional.
  - Separa el diseño del código lógico.
  - Facilita el desarrollo y la iteración sin recargar manualmente.
- 

## Requerimientos de instalación (`requirements.txt`)

In [...]

```
streamlit==1.32.2
numpy==1.25.2
pandas==2.1.4
matplotlib==3.7.5
scikit-learn==1.3.2
tensorflow==2.13.0
pillow==10.1.0
pydot==1.4.2
```

## Otros (Archivos generados en Proyecto 23)

historial\_entrenamiento\_23.pkl

modelo\_coloreado.dot

modelo\_coloreado.png

modelo\_plot.png

modelo23\_pesos\_weights.h5

modelo23.keras

prueba.dot

prueba.png

tokenizer\_23.pkl

## Conclusiones

Por medio de Streamlit o Flask, podemos visualizar el despliegue de nuestro clasificador de texto, mostrando los gráficos y métricas entregadas en cada entrenamiento y cada prueba. Este no es más que el proyecto 23 alojado en un servidor gratuito, donde no utilizamos los jupyter notebook, sino archivos .py.

---