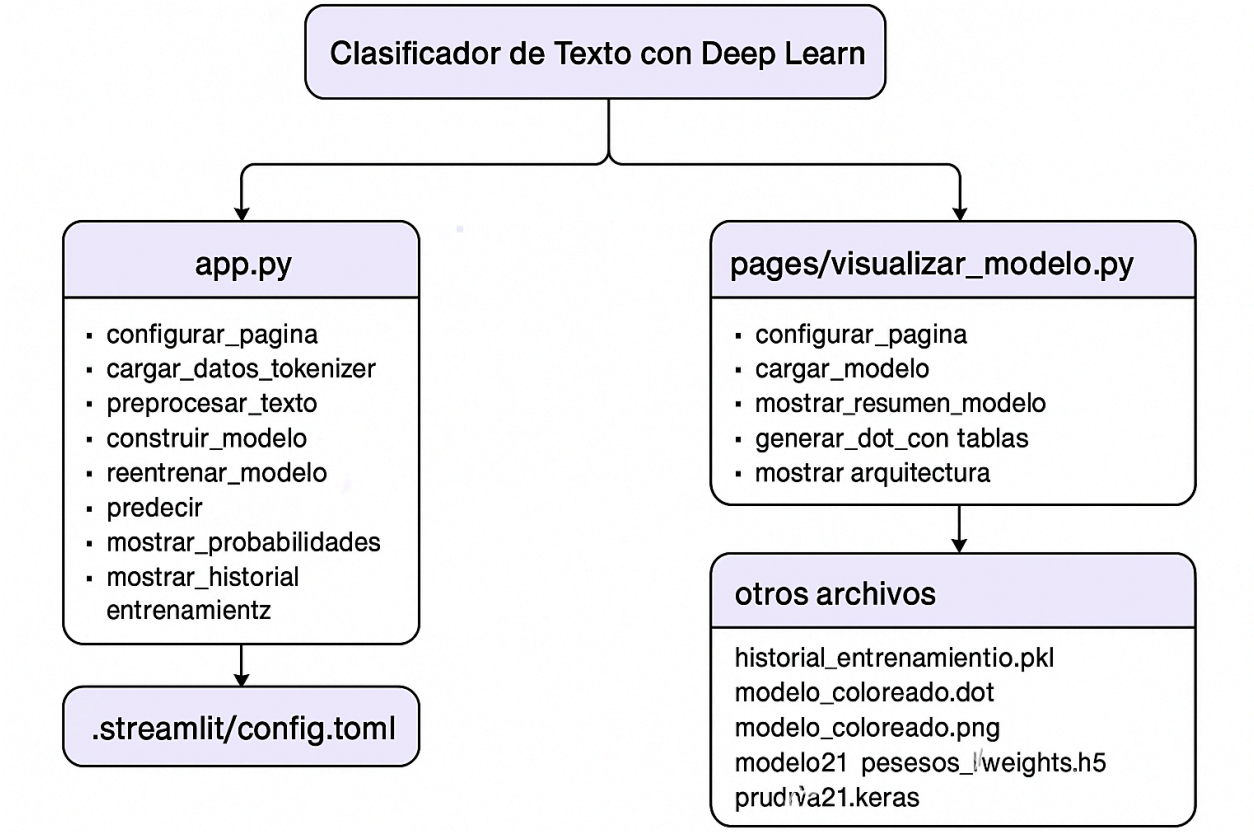
# Anexo 22

## 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:

🔗 **cnn-classicator-app.streamlit.app**




## Aplicación Principal (app.py)

In [...]

```python
# === 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
```

```python
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)
```

```python
# === 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("ℹ️ 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
```

```python
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)
```

```python
# === 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 […

```python
# === 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 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;
```

```
                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.tolm)

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