

Anexo 25

Proyecto 25: Generador de Resumen BiLSTM en Streamlit

Mg. Luis Felipe Bustamante Narváez

Este proyecto es el **Proyecto Final**, en el cuál diseñamos el modelo con **RNN's** implementado en **Streamlit**, a partir de la generación del archivo **.py**, para su correcta ejecución y despliegue, el cual se encuentra en el enlace:

 bilstm-app.streamlit.app

Arquitectura del Proyecto

```
resumen_bilstm/
├── .streamlit/
│   └── config.toml
├── data/
│   ├── df_total.csv
│   └── metricas_entrenamiento_25.csv
├── images/
│   ├── modelo_coloreado.dot
│   ├── modelo_coloreado.png
│   └── modelo_resumen_bilstm_25.png
├── models/
│   ├── historial_entrenamiento_25.pkl
│   ├── modelo_resumen_bilstm_25.keras
│   └── tokenizer_resumen_25.pkl
├── pages/
│   ├── __init__.py
│   ├── entrenamiento_modelo_25.py
│   ├── resumen_modelo_25.py
│   ├── resumir_texto_manual_25.py
│   └── visualizar_modelo_25.py
├── utils/
│   └── utils.py
├── app_resumen_25.py
└── requirements.txt``
```

El contenido de las carpetas, **models** e **images** se generan una vez se ejecute el **streamlit run app_resumen_25.py**

.streamlit (config.toml)

```
In [...] [theme]
base="light"
primaryColor="#e42ae3"
secondaryBackgroundColor="#c5b3d4"
textColor="#4c065e"
```

```
[server]
runOnSave = true

[client]
showSidebarNavigation = false

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

pages/

entrenamiento_modelo_25.py

```
In [... import streamlit as st
import pandas as pd
import numpy as np
import nltk
import pickle
import os
import nltk

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.models import Sequential
from tensorflow.keras.layers import Embedding, Bidirectional, LSTM, Dense
from tensorflow.keras.models import save_model
from tensorflow.keras.callbacks import Callback
from nltk.corpus import stopwords
import re

#st.set_page_config(page_title="Resumen de Noticias", layout="wide")

from utils.utils import mostrar_firma_sidebar

nltk.download('punkt')
nltk.download('stopwords')
stop_words = set(stopwords.words('spanish'))

st.title("📰 Reentrenar modelo BiLSTM para resumen de noticias")

def sidebar():
    with st.sidebar:
        st.header("🚀 Entrenamiento del modelo")
        st.markdown('---')

        st.subheader("📊 Métricas del Modelo")

        metrics_path = "data/metricas_entrenamiento_25.csv"
        if os.path.exists(metrics_path):
            metrics_df = pd.read_csv(metrics_path)
            #st.info(metrics_df)
```

[illegible]

```

random_state=42
)

# Modelo
modelo = Sequential()
modelo.add(Embedding(input_dim=num_words,
                     output_dim=128,
                     input_length=max_len))
modelo.add(Bidirectional(LSTM(64, return_sequences=False)))
modelo.add(Dense(1, activation='sigmoid'))

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

st.info("Entrenando modelo...")
# Barra de progreso y estado
progress_bar = st.progress(0)
status_text = st.empty()

class StreamlitProgressCallback(Callback):
    def __init__(self, epochs):
        super().__init__()
        self.epochs = epochs

    def on_epoch_end(self, epoch, logs=None):
        progress = int((epoch + 1) / self.epochs * 100)
        progress_bar.progress(progress)
        status_text.text(f"Época {epoch + 1}/{self.epochs} completada")
history = modelo.fit(
    X_train,
    y_train,
    epochs=epochs,
    batch_size=batch_size,
    validation_data=(X_test, y_test),
    verbose=0,
    callbacks=[StreamlitProgressCallback(epochs)]
)

st.success("Entrenamiento finalizado ✅")

# Mostrar resultados
df_hist = pd.DataFrame(history.history)
st.subheader("📊 Métricas del entrenamiento")
st.line_chart(df_hist[['loss', 'val_loss']])
st.line_chart(df_hist[['accuracy', 'val_accuracy']])

# Guardar métricas temporalmente
df_hist.to_csv("data/metricas_entrenamiento_25.csv", index=False)
modelo_g = save_model(modelo, "models/modelo_resumen_bilstm_25.keras")
with open("models/tokenizer_resumen_25.pkl", "wb") as f:
    pickle.dump(tokenizer, f)
with open("models/historial_entrenamiento_25.pkl", "wb") as g:
    pickle.dump(history.history, g)

if st.button('✅ Aceptar'):
    st.rerun()

```

```

    return modelo

# Parámetros
num_words = 10000
max_len = 40
batch_size = 32
sidebar()

# Epochs por slider
epochs = st.slider("Selecciona la cantidad de épocas",
                    min_value=1,
                    max_value=20,
                    value=5
                    )

if st.button("🚀 Entrenar modelo"):
    entrenamiento(epochs)

```

resumen_modelo_25.py

```

In [... import nltk
nltk.download('punkt')
from tensorflow.keras.preprocessing.sequence import pad_sequences
import numpy as np
from pages.entrenamiento_modelo_25 import limpiar_y_filtrar

def resumir_noticia(noticia, modelo, tokenizer, umbral, max_oraciones, max_len=50):
    # Dividir en oraciones y limpiar
    oraciones = [s.strip() for s in nltk.sent_tokenize(noticia, language='spanish') \
                  if len(s.strip()) > 0]

    # Truncar si hay demasiadas oraciones
    if len(oraciones) > max_oraciones:
        oraciones = oraciones[:max_oraciones]

    # Preprocesar oraciones
    X_filtrado = [limpiar_y_filtrar(oracion) for oracion in oraciones]

    # Tokenizar y rellenar
    secuencias = tokenizer.texts_to_sequences(oraciones)
    padded = pad_sequences(secuencias,
                           maxlen=max_len,
                           padding='post',
                           truncating='post'
                           )

    # Predecir con batch_size
    predicciones = modelo.predict(padded, batch_size=32, verbose=0)

    # Seleccionar oraciones relevantes
    resumen = [oraciones[i] for i in range(len(oraciones)) if predicciones[i] >= umbral]

    return resumen, X_filtrado

```

resumir_texto_manual_25.py

```

In [... import streamlit as st
import pickle
from tensorflow.keras.models import load_model
from wordcloud import WordCloud
import matplotlib.pyplot as plt

st.set_page_config(page_title="Resumen de Noticias", layout="wide")

from pages.resumen_modelo_25 import resumir_noticia
from utils.utils import mostrar_firma_sidebar

st.title("📄 Ingresar texto para resumir")

# ----- Sidebar -----
with st.sidebar:
    st.header("⚙️ Parámetros")
    umbral = st.slider("Ajustar umbral de relevancia",
                        min_value=0.1,
                        max_value=0.9,
                        value=0.5,
                        step=0.05
                      )
    max_oraciones = st.slider("Limitar número de oraciones a procesar",
                              min_value=10,
                              max_value=200,
                              value=100,
                              step=10
                             )

    st.page_link('app_resumen_25.py', label='**🏠 Página Principal**')
    st.page_link('pages/entrenamiento_modelo_25.py', label='**🚀 Reentrenamiento**')
    st.page_link('pages/visualizar_modelo_25.py', label='**🏗️ Arquitectura**')
    mostrar_firma_sidebar()

# ----- Cargar modelo y tokenizer -----
modelo = load_model("models/modelo_resumen_bilstm_25.keras", compile=False)

with open("models/tokenizer_resumen_25.pkl", "rb") as f:
    tokenizer = pickle.load(f)

# ----- Entrada de texto -----
st.subheader("✍️ Escribe o pega un texto")
texto_usuario = st.text_area("Introduce aquí la noticia o el texto\
                               completo que deseas resumir:", height=300)

if st.button("🔥 Generar Resumen"):
    if texto_usuario.strip():
        resumen, palabras_clave = resumir_noticia(texto_usuario,
                                                    modelo,
                                                    tokenizer,
                                                    umbral,
                                                    max_oraciones
                                                  )

        st.subheader("📄 Resumen Generado")
        if resumen:
            for i, oracion in enumerate(resumen, 1):
                st.markdown(f"{oracion}")

```

```

else:
    st.warning("⚠️ No se encontraron oraciones relevantes para este umbral.")

# ----- WordCloud -----
if palabras_clave:
    st.subheader("🕒 WordCloud del Resumen")
    texto_resumen = " ".join(palabras_clave)
    wc = WordCloud(background_color='white',
                    width=800,
                    height=400
                    ).generate(texto_resumen)

    fig, ax = plt.subplots(figsize=(10, 5))
    ax.imshow(wc, interpolation='bilinear')
    ax.axis("off")
    st.pyplot(fig)

else:
    st.error("❌ Por favor ingresa un texto antes de generar el resumen.")

```

visualizador_modelo_25.py

```

In [... from contextlib import redirect_stdout
import pydot
import streamlit as st
import pandas as pd
import matplotlib.pyplot as plt
from tensorflow.keras.models import load_model
from tensorflow.keras.utils import plot_model
import io
import os
import time
import numpy as np

st.set_page_config(page_title="Resumen de Noticias", layout="wide")

from pages.entrenamiento_modelo_25 import entrenamiento
from utils.utils import mostrar_firma_sidebar

st.title("Visualización del Modelo BiLSTM para Resumen de Textos")

def sidebar():
    with st.sidebar:
        st.header("🏠 Arquitectura del modelo")
        st.markdown('---')
        st.page_link('app_resumen_25.py', label='**🏠 Página Principal**')
        st.page_link('pages/entrenamiento_modelo_25.py', label='**🚀 Reentrenamiento**')
        st.page_link('pages/resumir_texto_manual_25.py', label='**📄 Tu resumen**')

sidebar()

def graficas(metricas):
    st.header("Gráfica de Pérdida")
    fig1, ax1 = plt.subplots()
    ax1.plot(metricas['loss'], label='Entrenamiento')
    ax1.plot(metricas['val_loss'], label='Validación')

```

```

ax1.set_xlabel("Épocas")
ax1.set_ylabel("Pérdida")
ax1.set_title("Historial de Pérdida")
ax1.legend()
st.pyplot(fig1)

st.header("Gráfica de Precisión")
fig2, ax2 = plt.subplots()
ax2.plot(metricas['accuracy'], label='Entrenamiento')
ax2.plot(metricas['val_accuracy'], label='Validación')
ax2.set_xlabel("Épocas")
ax2.set_ylabel("Precisión")
ax2.set_title("Historial de Precisión")
ax2.legend()
st.pyplot(fig2)

# === 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_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.png",
            mime="image/png"
        )

def cargar_modelo(ruta):
    # Cargar el modelo

```

```

try:
    model = load_model("models/modelo_resumen_bilstm_25.keras")
    return model
except Exception as e:
    st.error(f"Error al cargar el modelo: {e}")
    st.stop()

# === Interfaz Principal ===
def main():
    # Cargar el historial
    if os.path.exists("data/metricas_entrenamiento_25.csv"):
        metricas = pd.read_csv("data/metricas_entrenamiento_25.csv")
        graficas(metricas)
    else:
        st.error("No se encontró el archivo 'data/metricas_entrenamiento_25.csv'")
        st.stop()
    modelo = cargar_modelo("models/modelo_resumen_bilstm_25.keras")
    mostrar_resumen_modelo(modelo)
    mostrar_arquitectura(modelo,
                          "images/modelo_coloreado.dot",
                          "images/modelo_coloreado.png"
                          )

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

```

utils/

utils.py

In [... `import streamlit as st`

```

# pie de página
def mostrar_firma_sidebar():
    st.sidebar.markdown("""
        <style>
            .firma-sidebar {
                position: fixed;
                bottom: 20px;
                left: 13px;
                width: 10pts;
                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;
            }
        </style>
    """)

```

```

        .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"><img alt="GitHub icon" data-bbox="705 175 720 190"/> GitHub</a> ·
        <a href="https://www.linkedin.com/in/lfbn2" target="_blank"><img alt="LinkedIn icon" data-bbox="755 190 770 205"/> LinkedIn</a>
    </div>
    """, unsafe_allow_html=True)

```

Aplicación Principal

app_resumen_25.py

```

In [... import streamlit as st
import pandas as pd
from wordcloud import WordCloud
import matplotlib.pyplot as plt
from tensorflow.keras.models import load_model
import pickle
import os

st.set_page_config(page_title="Resumen de Noticias", layout="wide")

#aquí porque esa página usa streamlit y el set debe quedar primero.
from pages.resumen_modelo_25 import resumir_noticia
from utils.utils import mostrar_firma_sidebar

st.title("🧠 Generador de Resúmenes con BiLSTM")

# Cargar modelo y métricas
modelo = load_model("models/modelo_resumen_bilstm_25.keras")

# Cargar tokenizer
with open("models/tokenizer_resumen_25.pkl", "rb") as f:
    tokenizer = pickle.load(f)

# Cargar dataset
df = pd.read_csv("data/df_total.csv", encoding='utf-8')
# asumimos que no hay columna título
df['titulo'] = df['news'].apply(lambda x: x.split('.')[0][:100])

with st.sidebar:
    st.header("🔍 Selecciona una noticia")
    titulo_seleccionado = st.selectbox("Títulos disponibles:", df['titulo'])
    umbral = st.slider("Ajustar umbral de relevancia",
                        min_value=0.1,
                        max_value=0.9,
                        value=0.5,
                        step=0.05
                    )
    max_oraciones = st.slider("Limitar número de oraciones a procesar",

```

```

        min_value=10,
        max_value=200,
        value=100,
        step=10
    )

def sidebar():
    with st.sidebar:
        st.header("🔍 Entrenamiento del modelo")
        st.markdown('---')

        st.subheader("📊 Métricas del Modelo")

        metrics_path = "data/metricas_entrenamiento_25.csv"
        if os.path.exists(metrics_path):
            metrics_df = pd.read_csv(metrics_path)
            #st.info(metrics_df)
            #st.line_chart(metrics_df[['loss', 'val_loss']])
            st.line_chart(metrics_df[['accuracy', 'val_accuracy']])
        else:
            st.info("No se encontraron métricas guardadas.")
            st.markdown('---')
            st.page_link('pages/entrenamiento_modelo_25.py', label='** 🚀 Reentrenamiento**')
            st.page_link('pages/visualizar_modelo_25.py', label='** 🏗️ Arquitectura**')
            st.page_link('pages/resumir_texto_manual_25.py', label='** 📄 Tu resumen**')

mostrar_firma_sidebar()

# ----- Selección de noticia -----

# Obtener la noticia seleccionada
noticia = df[df['titulo'] == titulo_seleccionado]['news'].values[0]

st.subheader("📰 Noticia Original")
st.write(noticia)

sidebar()

# ----- Generar resumen -----
if st.button("🔥 Generar Resumen"):
    resumen, word = resumir_noticia(noticia, modelo, tokenizer, umbral, max_oraciones)
    st.subheader("📄 Resumen Generado")
    if resumen:
        for i, oracion in enumerate(resumen, 1):
            st.markdown(f"{oracion}\n\n")
    else:
        st.warning("⚠️ No se encontraron oraciones relevantes para este umbral.")

# ----- WordCloud del resumen -----
if word:
    st.subheader("☁️ WordCloud del Resumen")
    resumen_texto = " ".join(word)
    wc = WordCloud(background_color='white',
                    width=800,
                    height=400
                    ).generate(resumen_texto)

```

```
fig, ax = plt.subplots(figsize=(10, 5))
ax.imshow(wc, interpolation='bilinear')
ax.axis("off")
st.pyplot(fig)
```

requirements.txt

```
In [... streamlit==1.32.2
numpy==1.26.4
pandas==2.1.4
matplotlib==3.7.5
scikit-learn==1.3.2
tensorflow==2.17.1
pillow==10.1.0
pydot==1.4.2
graphviz==0.20.1
protobuf==3.20.*
setuptools>=68.0.0
nltk==3.8.1
wordcloud==1.9.3
```

Conclusiones

Por medio de Streamlit o Flask, podemos visualizar el despliegue de nuestro generador de resúmenes, mostrando los gráficos y métricas entregadas en cada entrenamiento y cada prueba. Además de poder reentrenar el modelo y observar nuevas métricas directamente desde el entorno web. Este proyecto está alojado en un servidor gratuito, donde no utilizamos los jupyter notebook, sino archivos .py.

Mg. Luis Felipe Bustamante Narváez