

Índice de Anexos

Ineligencia Artificial 1

Mg. Luis Felipe Bustamante Narváez

Módulo 1

- Proyectos del 1 al 7

Anexo 1:

Tokenization, Stimming and Lemmatization

Anexo 2

Aplicación de TF-IDF

Anexo 3

Aplicación de Words Embedding

Anexo 4

Creación de Words Embedding Nivel 1

Anexo 5

Creación de Words Embedding Nivel 2

Anexo 6

Conversión masiva de nombres de archivos

Anexo 7

Creación de Words Embedding Nivel 3

Módulo 2

- Proyectos del 8 al 10

Anexo 8

Clasificador de texto

Anexo 9

Generador de texto

Anexo 10

Spinning de Texto

Módulo 3

- Proyectos del 11 al 16

Anexo 11

Detector de Spam

Anexo 12

Analisis de Sentimiento

Anexo 13

Analisis de Sentimiento Multiclasificación

Anexo 14

Resumen de textos con Vectorización

Anexo 15

Resumen de textos con Text Rank

Anexo 16

Modelado de Temas con LDA

Módulo 4

- Proyectos del 17 al 24

Anexo 17

Regresión Lineal con TensorFlow

Anexo 18

Ánalisis de Sentimiento con TensorFlow

Anexo 19

Creación de Red Neuronal

Anexo 20

Manejo de TensorFlow para NLP

Anexo 21

Clasificador de Texto con CNN's para NLP

Anexo 22

Clasificador de Texto CNN en Streamlit

Anexo 23

Clasificador de Texto con RNN's para NLP

Anexo 24

Clasificador de Texto RNN en Streamlit

Módulo 5

- Proyecto 25

Anexo 25

Generador de Resumen BiLSTM en Streamlit

Anexo 1

Proyecto 1: Tokenization, Stimming and Lemmatization

Mg. Luis Felipe Bustamante Narváez

Tokenization

```
In [... # Instalamos sklearn  
!pip install -U scikit-learn  
  
In [... import pandas as pd  
  
In [... # Creamos el dataframe para mostrar la base de datos  
df = pd.read_csv('df_total.csv', encoding='UTF-8')  
  
In [... df  
  
In [... # Mostramos la estructura de los datos  
df['news'][2]  
  
In [... # Separamos los datos en variable independiente y variable dependiente  
# variable independiente  
X = df['news']  
# variable dependiente (A predecir)  
Y = df['Type']  
  
In [... # Mostramos la variable dependiente  
df['Type'].value_counts()  
  
In [... from sklearn.model_selection import train_test_split  
  
In [... # Asignamos los datos de entrenamiento (80%) y prueba (20%)  
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2)  
  
In [... # Mostramos los datos de entrenamiento (1217datos*0.8 = 973)  
X_train  
  
In [... # Mostramos los datos de prueba (1217datos*0.2 = 244)  
X_test  
  
In [... # Mostramos los datos a predecir de entrenamiento (1217datos*0.8 = 973)  
Y_train  
  
In [... # Mostramos los datos a predecir de prueba (1217datos*0.2 = 244)  
Y_test  
  
In [... # Vectorizamos los datos  
from sklearn.feature_extraction.text import CountVectorizer  
vectorizer = CountVectorizer()  
  
In [... # vectorizamos los datos de entrenamiento  
X_train_transformed = vectorizer.fit_transform(X_train)  
# vectorizamos los datos de prueba  
X_test_transformed = vectorizer.transform(X_test)  
  
In [... # Mostramos los datos transformados entrenamiento (de forma nativa)  
X_train_transformed
```

```
In [...] #Mostramos Los datos transformados entrenamiento
X_train_transformed_dense = X_train_transformed.toarray()
X_train_transformed_dense

In [...] #Mostramos Los datos transformados prueba (de forma nativa)
X_test_transformed

In [...] #Mostramos Los datos transformados prueba
X_test_transformed_dense = X_test_transformed.toarray()
X_test_transformed_dense

In [...] from sklearn.naive_bayes import MultinomialNB
from sklearn import metrics

In [...] # Creamos La variable para el modelo
model = MultinomialNB()
# Entrenamos el modelo con Los datos de entrenamiento vectorizados y Los valores reales dependientes
model.fit(X_train_transformed, Y_train)
#Creamos La predicción con Los datos de prueba vectorizados
Y_predict = model.predict(X_test_transformed)
#Revisamos el porcentaje de predicción
accuracy = metrics.accuracy_score(Y_test, Y_predict)
print(accuracy)
```

Stimming

```
In [...] import nltk
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import SnowballStemmer

In [...] nltk.download('punkt')
nltk.download('stopwords')

In [...] stemmer = SnowballStemmer('spanish')

In [...] # Función para tokenizar y stimmer
def tokenize_and_stem(text):
    tokens = word_tokenize(text.lower())
    stems = [stemmer.stem(token) for token in tokens if token.isalpha()]
    return ' '.join(stems)

# Otra manera de mostrar esta función
"""

def tokenize_and_stem(text):
    # Tokenizar el texto y convertirlo a minúsculas
    tokens = word_tokenize(text.lower())

    # Aplicar stemming con un bucle for
    stems = []
    for token in tokens:
        if token.isalpha(): # Filtrar solo palabras (sin números ni símbolos)
            stem = stemmer.stem(token) # Aplicar stemming
            stems.append(stem) # Agregar a la lista

    # Unir los stems en un solo string y retornarlo
    return ' '.join(stems)
"""
```

```
In [...] # Agregamos una nueva columna al dataframe para mostrar Los tokens
df['news_stemmer'] = df['news'].apply(tokenize_and_stem)

In [...] df

In [...] # Separamos Los datos en variables de entrada y etiquetas
Xs = df['news_stemmer']
Ys = df['Type']
# Creamos el entrenamiento
Xs_train, Xs_test, Ys_train, Ys_test = train_test_split(Xs, Ys, test_size=0.2)
vectorizerS = CountVectorizer()
# Transformamos Los documentos en vectores
Xs_train_transformed = vectorizerS.fit_transform(Xs_train)
Xs_test_transformed = vectorizerS.transform(Xs_test)
# Creamos el modelo
modelS = MultinomialNB()
# Entrenamos el modelo
modelS.fit(Xs_train_transformed, Ys_train)
# Medimos el rendimiento del modelo
Ys_predict = modelS.predict(Xs_test_transformed)
# Mostramos su eficiencia
accuracyS = metrics.accuracy_score(Ys_test, Ys_predict)
print(accuracyS)
```

Comparamos las dimensiones del texto original y los stemming

```
In [...] origin = X_train_transformed.shape
stimming = Xs_train_transformed.shape

# Opcional --- creamos un DataFrame para mejorar La visual y repasar
dfCompare = pd.DataFrame({
    'DataSet': ['X_train_transformed', 'Xs_train_transformed'],
    'Vectores': [origin[0], stimming[0]],
    'Dimensiones': [origin[1], stimming[1]]
})
dfCompare
```

Lemmatization

```
In [...] import spacy

In [...] nlp = spacy.load('es_core_news_sm')

In [...] # función de Lemmatization
def lemmatize_text(text):
    doc = nlp(text.lower())
    lemmas = [token.lemma_ for token in doc if token.is_alpha]
    return ' '.join(lemmas)

In [...] # Agregamos una nueva columna al dataframe para mostrar Los Lemmas
df['news_lemma'] = df['news'].apply(lemmatize_text)

In [...] df['news_lemma'][5]

In [...] # Separamos Los datos en variables de entrada y etiquetas
Xl = df['news_lemma']
Yl = df['Type']
# Creamos el entrenamiento
Xl_train, Xl_test, Yl_train, Yl_test = train_test_split(Xl, Yl, test_size=0.2)
vectorizerL = CountVectorizer()
```

```

# Transformamos los documentos en vectores
Xl_train_transformed = vectorizerL.fit_transform(Xl_train)
Xl_test_transformed = vectorizerL.transform(Xl_test)
# Creamos el modelo
modelL = MultinomialNB()
# Entrenamos el modelo
modelL.fit(Xl_train_transformed, Yl_train)
# Medimos el rendimiento del modelo
Yl_predict = modelL.predict(Xl_test_transformed)
# Mostramos su eficiencia
accuracyL = metrics.accuracy_score(Yl_test, Yl_predict)
print(accuracyL)

```

Comparamos las dimensiones del texto original y los stemming

```

In [...]: origin = X_train_transformed.shape
          stemming = Xs_train_transformed.shape
          lemmatizing = Xl_train_transformed.shape

# Opcional --- creamos un DataFrame para mejorar la visual y repasar
dfCompare = pd.DataFrame({
    'DataSet': ['X_train_transformed', 'Xs_train_transformed', 'Xl_train_transformed'],
    'Vectores': [origin[0], stemming[0], lemmatizing[0]],
    'Dimensiones': [origin[1], stemming[1], lemmatizing[1]]})
dfCompare

```

Conclusiones

Se puede tomar la decisión de seleccionar el mejor modelo, aunque los datos son pocos, se puede observar una buena predicción, aunque en el texto se ve mejor la lemmatización, este requiere un gasto computacional más alto, que no se ve reflejado, para este ejemplo, en la eficacia del modelo.

Mg. Luis Felipe Bustamante Narváez

Anexo 2

Proyecto 2: Aplicación de TF-IDF

Mg. Luis Felipe Bustamante Narváez

Importamos los datos para su manipulación

```
In [... import pandas as pd  
df = pd.read_csv('movie_metadata.csv', encoding='UTF-8')  
df.head()  
  
Out[... color director_name num_critic_for_reviews duration director_facebook_likes actor_3_facebook_likes actor_2_name actor_1_facebook_likes gross genres ... num_user_for_rev...  
0 Color James Cameron 723.0 178.0 0.0 855.0 Joel David Moore 1000.0 760505847.0 Action|Adventure|Fantasy|Sci-Fi ... 30:  
1 Color Gore Verbinski 302.0 169.0 563.0 1000.0 Orlando Bloom 40000.0 309404152.0 Action|Adventure|Fantasy ... 12:  
2 Color Sam Mendes 602.0 148.0 0.0 161.0 Rory Kinnear 11000.0 200074175.0 Action|Adventure|Thriller ... 9:  
3 Color Christopher Nolan 813.0 164.0 22000.0 23000.0 Christian Bale 27000.0 448130642.0 Action|Thriller ... 27:  
4 NaN Doug Walker NaN NaN 131.0 NaN Rob Walker 131.0 NaN Documentary ... N  
5 rows × 28 columns
```

```
In [... # añadimos la columna id al principio de la tabla  
df.insert(0, 'id', range(1, len(df)+ 1))  
df
```

```
In [... # Mostramos las columnas que no se ven para poder seleccionarlas  
df.iloc[:,10:20].head()
```

```
In [... # De acuerdo a esto, necesitamos las columnas id, genres, movie_title y plot_keywords  
# verificamos su ubicación  
df.columns
```

```
In [... # Necesitamos las columnas: 0, 10,12, 17  
cols_mantener = [0, 10, 12, 17]  
df_new = df.iloc[:,cols_mantener]  
df_new
```

```
In [... # cambiamos los nombres de las columnas para poder utilizarlos de mejor manera  
df_new = df_new.rename(columns={df_new.columns[1]:'genero',  
                               df_new.columns[2]:'titulo',  
                               df_new.columns[3]:'plot_keywords'})  
df_new
```

```
In [... # necesitamos reordenar las columnas para el manejo de los datos  
col_mover = df_new.pop('titulo')  
df_new.insert(1, 'titulo', col_mover)  
df_new
```

```
In [... # eliminamos los espacios en blanco al principio y al final de cada texto  
for i in range(1, len(df_new.columns)):  
    df_new.iloc[:,i] = df_new.iloc[:,i].str.strip()  
df_new
```

```
In [... # reemplazamos el carácter | por espacio vacío  
for i in range(2, len(df_new.columns)):  
    df_new.iloc[:,i] = df_new.iloc[:,i].str.replace('|', ' ')  
df_new.head()
```

	id	titulo	genero	plot_keywords	texto
0	1	Avatar	Action Adventure Fantasy Sci-Fi	avatar future marine native paraplegic	Action Adventure Fantasy Sci-Fi avatar future ...
1	2	Pirates of the Caribbean: At World's End	Action Adventure Fantasy	goddess marriage ceremony marriage proposal pi...	Action Adventure Fantasy goddess marriage cere...
2	3	Spectre	Action Adventure Thriller	bomb espionage sequel spy terrorist	Action Adventure Thriller bomb espionage seque...
3	4	The Dark Knight Rises	Action Thriller	deception imprisonment lawlessness police off...	Action Thriller deception imprisonment lawless...
4	5	Star Wars: Episode VII - The Force Awakens	Documentary	NaN	Documentary nan

```
In [... # Unimos las columnas genero y plot_keywords
df_new['texto'] = df_new[['genero', 'plot_keywords']].apply(lambda row: ' '.join(row.values.astype(str)), axis=1)
df_new
```

```
In [... # observamos con un ejemplo como quedó la columna texto
df_new['texto'].iloc[0]
```

TF-IDF

```
In [... from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity, euclidean_distances
```

```
In [... # vectorizamos poniendo un número máximo de tokens, entre más palabras haya, debe ser mayor este dato
tfIdf = TfidfVectorizer(max_features=3000)
```

```
In [... # Matriz de vectores (total de registros, total de dimensiones creadas)
X = tfIdf.fit_transform(df_new['texto'])
X
```

```
In [... # Creamos una serie para mapear las etiquetas de los nombres de las películas
movies = pd.Series(df_new.index, index=df_new['titulo'])
movies
```

```
In [... # Podemos buscar el índice de cada película
indice = movies['A Plague So Pleasant']
indice
```

```
In [... # consultamos
consulta = X[indice]
consulta
```

```
In [... # Observamos el vector
consulta2 = consulta.toarray()
consulta2
```

```
Out[... array([[0., 0., 0., ..., 0., 0., 0.]], shape=(1, 3000))
```

```
In [... # observemos las columnas del vector donde no hay datos en cero, donde hay tokens
print(consulta)
```

```
In [... # calculamos la similitud de los vectores (el valor de prueba con el resto de los datos)
simil = cosine_similarity(consulta2, X)
simil
```

```
Out[... array([[0.          , 0.          , 0.05807863, ... , 1.          , 0.64349593,
               0.          ],], shape=(1, 5043))
```

```
In [... # Lo anterior muestra el arreglo de similitudes, veamos un ejemplo de similitud entre
# La película 0 y nuestra consulta, y luego con la película 2 y nuestra consulta
comparar = 1763
a = simil[0][comparar] #simil[consulta (siempre es 0), peli a comparar]
print(f'''La similitud entre la película
"{df_new.iloc[comparar, 1]}" y nuestra película consultada,
"{df_new.iloc[indice, 1]}", es de {a}'''')
```

La similitud entre la película
 "Identity" y nuestra película consultada,
 "A Plague So Pleasant", es de 0.039206668970362867

```
In [... # calcular la similitud de manera más profesional
simil2 = simil.flatten()
```

```
In [... # ejemplo
comparar = 1763
```

```

b = simil2[comparar] #simil[consulta (siempre es 0), peli a comparar]
print(f'''La similitud entre la película
"{df_new.iloc[comparar, 1]}" y nuestra película consultada,
"{df_new.iloc[indice, 1]}", es de {b}''')
# Nótese que de esta manera solo se pone el valor de la película a comparar

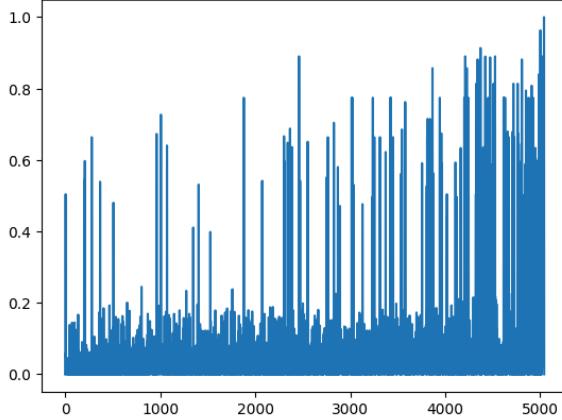
```

Gráficos de similitud

```

In [... !pip install matplotlib
In [... import matplotlib.pyplot as plt
In [... # Recordemos que usamos simil2 ya que esta está vectorizada con el flatten
plt.plot(simil2)
plt.show()

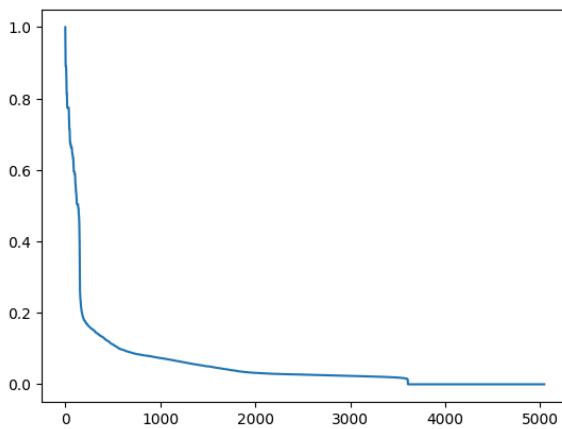
```



```

In [... # ordenemos la similitud
similOrden = (-simil2).argsort()
similOrden
In [... # graficamos ordenadamente de mayor a menor similitud
plt.plot(simil2[similOrden])
plt.show()
#Esto nos muestra una gráfia logarítmica generada por IDF

```



Recomendación de películas

```

In [... #Ordenamos la similitud omitiendo la película de la consulta
      #creando un top 10 de recomendaciones
recomendacion = (-simil2).argsort()[1:11]
In [... # Creamos la lista del top10 de recomendados
      df_new['titulo'].iloc[recomendacion]

```

```
Out[...]: 5002      Run, Hide, Die
        4372    A Beginner's Guide to Snuff
        4211      Antibirth
        4525    The Vatican Exorcisms
        2461      Restoration
        5028      Tin Can Man
        4422      Unsullied
        4473      Forget Me Not
        4341  The Curse of Downers Grove
        4807      Fabled
Name: titulo, dtype: object
```

Conclusiones

Al consultar una película cualquiera, como en el ejemplo, "A Plague So Pleasant", de acuerdo con el género y las plot_keywords de la tabla, podemos recomendar un top 10 de películas que tienen similitud con esta, algo básico, pero similar a lo que usan las aplicaciones de streaming como NetFlix, Disney, entre otras.

Mg. Luis Felipe Bustamante Narváez

Anexo 3

Proyecto 3: Aplicación de Words Embedding

Mg. Luis Felipe Bustamante Narváez

Analogías

```
In [... !pip install gensim
In [... import gensim
In [... #Cargamos el archivo de la db en un muestreo de vectores
vectores = gensim.models.KeyedVectors.load_word2vec_format('SBW-vectors-300-min5.txt')
In [... # Definimos la función que analizará la similitud
def analogics(v1, v2, v3):
    simil = vectores.most_similar(positive=[v1, v3], negative=[v2]) #buscamos el negativo
    print(f'{v1} es a {v2}, como {simil[0][0]} es a {v3}')
In [... # Llamamos la función con un ejemplo
analogics('rey', 'hombre', 'mujer')
rey es a hombre, como reina es a mujer
In [... analogics('Colombia', 'colombiano', 'venezolano')
Colombia es a colombiano, como Venezuela es a venezolano
In [... analogics('pan', 'trigo', 'leche')
pan es a trigo, como yogur es a leche
In [... analogics('vaca', 'leche', 'huevo')
vaca es a leche, como gallina es a huevo
In [... analogics('lápiz', 'borrador', 'corrector')
lápiz es a borrador, como pincel es a corrector
In [... analogics('chimenea', 'humo', 'agua')
chimenea es a humo, como desagüe es a agua
In [... # Observemos que no siempre es preciso
analogics('carne', 'carnívoro', 'vegetariano')
carne es a carnívoro, como carnes es a vegetariano
```

Valores cercanos

```
In [... #definimos la función de los cercanos
```

```
def cercanos(v):
    vecinos = vectores.most_similar(positive=[v])
    print(f'Vecinos de {v}:\n')
    for word, score in vecinos:
        print(f'{word}')
```

```
In [... cercanos('rey')
```

Vecinos de rey:

```
monarca
reyes
reino
príncipe
reina
Harthacnut
Ragnald
Sverkersson
regente
Hardeknut
```

```
In [... cercanos('ves')
```

Vecinos de ves:

```
sabes
piensas
dices
tú
verás
vas
sientes
estás
tienes
crees
```

```
In [... cercanos('vez')
```

Vecinos de vez:

```
ocasión
cuando
nuevamente
ya
temporada
rememoraría
luego
Súperfinal
que
cosa
```

```
In [... cercanos('vegetariano')
```

Vecinos de vegetariano:

vegetariana
vegano
vegetarianismo
vegetarianos
vegana
abstemio
veganismo
crudiveganismo
veganos
crudivorismo

Conclusiones

Con la base de datos consumida, se puede observar el comportamiento de los words embedding, cómo hallar similitudes entre palabras asociadas a un contexto y cómo visualizar las palabras que constituyen una clasificación asociada a una palabra específica.

Mg. Luis Felipe Bustamante Narváez

Anexo 4

Proyecto 4: Creación de Words Embedding Nivel 1

Mg. Luis Felipe Bustamante Narváez

Word2Vec

Es un modelo que se utiliza para aprender representaciones vectoriales de palabras. Estas representaciones pueden capturar muchas propiedades lingüísticas de las palabras, como su significado semántico, gramatical y hasta contextual.

Para este primer ejemplo, usaremos un texto corto llamado mundiales que tiene alrededor de 50.000 caracteres.

Librerías

```
In [...] !pip install pypdf2
```

```
In [...] import string
from gensim.models import Word2Vec
import PyPDF2
```

Cargamos el documento

```
In [...] with open('Entrenamiento_Word2Vec/mundiales.txt',
                  'r', encoding='utf-8') as file:
    documento = file.read()
```

```
In [...] documento
```

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

```
Out[...] 48155
```

Procesamiento de datos

El objetivo del procesamiento es convertir el documento en una lista de frases, y cada frase en una lista de palabras, eliminando signos de puntuación y convirtiendo todo a minúsculas.

```
In [...] # Dividimos el documento en frases usando La coma como separador
frases = documento.split(',')
len(frases)
```

```
Out[...] 533
```

```
In [...] # Mostramos un ejemplo
frases[0]
```

```
Out[...]: 'Capítulo 1: Historia del Mundial de Fútbol\nLos primeros pasos del Mundial\nLa historia del Mundial de Fútbol se remonta a principios del siglo XX'
```

```
In [...]: # Mostramos un ejemplo  
frases[500]
```

```
Out[...]: '**Adidas**'
```

```
In [...]: # Limpiamos las frases  
frases_limpias = []  
for frase in frases:  
    # Eliminamos la puntuación y dividimos por espacios  
    tokens = frase.translate(str.maketrans(' ', ' ', string.punctuation)).split()  
    # print(tokens) # para mostrar qué ha hecho hasta aquí  
    # Convertimos a minúsculas  
    tokens = [word.lower() for word in tokens]  
    # print(tokens) # para mostrar qué ha hecho hasta aquí  
    if tokens:  
        frases_limpias.append(tokens)
```

```
In [...]: # Mostramos los resultados  
frases_limpias[500]
```

```
Out[...]: ['adidas']
```

Entrenamiento del modelo Word2Vec

```
In [...]: model = Word2Vec(sentences=frases_limpias,  
                      vector_size=500,  
                      window=5,  
                      min_count=1,  
                      workers=4)
```

Explicación:

- sentences: Es la lista de palabras que vamos a vectorizar
- vector_size: Es el tamaño de dimensiones que le daremos al vector
- window: Son la cantidad de palabras por encima y por debajo que le darán contexto
- min_count: La aparición mínima de una palabra para tenerla en cuenta en el entrenamiento
- workers: Cantidad de núcleo de procesador que vamos a invertir en el entrenamiento

Pruebas

```
In [...]: # Verificamos el vector para alguna palabra  
vector = model.wv['mundial']  
vector
```

```
In [...]: # Mostramos las palabras cercanas  
palabras_cercanas = model.wv.most_similar('jugador', topn=10)  
palabras_cercanas  
# Es probable que la similitud falle por tener tan pocas palabras en el texto
```

```
Out[...]: [('historia', 0.9089663624763489),  
          ('su', 0.9083009362220764),  
          ('con', 0.9079578518867493),  
          ('y', 0.9078695774078369),  
          ('brasil', 0.9078560471534729),  
          ('en', 0.9078254699707031),  
          ('más', 0.9077353477478027),  
          ('del', 0.9077184200286865),  
          ('francia', 0.9076839685440063),  
          ('juego', 0.9075537919998169)]
```

Guardar modelo

```
In [...]: model.save('Entrenamiento_Word2Vec/mundiales.model')
```

Cargar el modelo

```
In [...]: modelo_cargado = Word2Vec.load('Entrenamiento_Word2Vec/mundiales.model')
```

```
In [...]: # Probamos con el modelo cargado  
palabras_cercanas2 = modelo_cargado.wv.most_similar('jugador', topn=10)  
palabras_cercanas2
```

```
Out[...]: [('historia', 0.9089663624763489),  
          ('su', 0.9083009362220764),  
          ('con', 0.9079578518867493),  
          ('y', 0.9078695774078369),  
          ('brasil', 0.9078560471534729),  
          ('en', 0.9078254699707031),  
          ('más', 0.9077353477478027),  
          ('del', 0.9077184200286865),  
          ('francia', 0.9076839685440063),  
          ('juego', 0.9075537919998169)]
```

Guardar Embedido

Existen dos maneras, usando .txt sin binarios, y usando .bin con binarios.

```
In [...]: model.wv.save_word2vec_format('Entrenamiento_Word2Vec/munidiales_emb.txt', binary=False)  
model.wv.save_word2vec_format('Entrenamiento_Word2Vec/munidiales_emb.bin', binary=True)
```

Cargar Embedidos

Si se carga el .txt, se usa sin binarios; si se carga el .bin, se usa con binarios

```
In [...]: from gensim.models import KeyedVectors  
embedding_cargado_txt = KeyedVectors.load_word2vec_format(  
    'Entrenamiento_Word2Vec/munidiales_emb.txt', binary=False)
```

```
In [...]: embedding_cargado_bin = KeyedVectors.load_word2vec_format(  
    'Entrenamiento_Word2Vec/munidiales_emb.bin', binary=True)
```

```
In [...]: # Probamos
```

```
embedding_cargado_txt
```

```
Out[...]: <gensim.models.keyedvectors.KeyedVectors at 0x21302849a30>
```

```
In [...]: # Probamos
```

```
embedding_cargado_bin
```

```
Out[...]: <gensim.models.keyedvectors.KeyedVectors at 0x213062ccaa0>
```

Analogías

```
In [...]:
```

```
def analogics(v1, v2, v3):
    simil = embedding_cargado_bin.most_similar(positive=[v1,v3],
                                                negative=[v2])
    print(f'{v1} es a {v2}, como {simil[0][0]} es a {v3}')
```

```
In [...]:
```

```
analogics('jugador', 'fútbol', 'historia')
```

```
jugador es a fútbol, como brasil es a historia
```

Conclusiones

El texto mundiales tiene cerca de 50.000 caracteres, lo que implica una base de datos pequeña para entrenar un modelo. De cierta forma, el modelo se ajusta en algunos casos puntuales, pero suele mostrar demasiadas stopwords, que tendríamos que manipular para mejorar la predicción de analogías. Veremos con un texto más grande, como se generaría la predicción, por ejemplo el libro "Cien años de soledad" de Gabriel García Márquez.

Mg. Luis Felipe Bustamante Narváez

Anexo 5

Proyecto 5: Creación de Words Embedding Nivel 2

Mg. Luis Felipe Bustamante Narváez

Word2Vec

Es un modelo que se utiliza para aprender representaciones vectoriales de palabras. Estas representaciones pueden capturar muchas propiedades lingüísticas de las palabras, como su significado semántico, gramatical y hasta contextual.

Para este segundo ejemplo, usaremos un texto más extenso llamado "Cien años de soledad" del escritor colombiano Gabriel García Márquez, que tiene alrededor de 139.318 palabras y 823735 caracteres.

Librerías

```
In [...] !pip install pypdf2
Requirement already satisfied: pypdf2 in c:\users\luis_\anaconda3\envs\notebook\lib\site-packages (3.0.1)

In [...] import string
from gensim.models import Word2Vec
import PyPDF2
```

Cargamos el documento

```
In [...] def extraer_texto_desde_pdf(ruta_archivo):
    with open(ruta_archivo, 'rb') as archivo:
        lector = PyPDF2.PdfReader(archivo)
        texto = ''
        for pagina in range(len(lector.pages)):
            texto += lector.pages[pagina].extract_text()
    return texto

In [...] documento = extraer_texto_desde_pdf('Entrenamiento_Word2Vec/cien_anos_de_soledad.pdf')

In [...] len(documento)

Out[...] 823735
```

Procesamiento de datos

El objetivo del procesamiento es convertir el documento en una lista de frases, y cada frase en una lista de palabras, eliminando signos de puntuación y convirtiendo todo a minúsculas.

```
In [...] # Dividimos el documento en frases usando la coma como separador
frases = documento.split(',')
len(frases)

Out[...] 8854

In [...] # Mostramos un ejemplo
frases[500]

Out[...] ' las huestes de su padre tenían un aspecto de náufragos sin escapatoria'
```

```
In [... # Mostramos un ejemplo
frases[3000]

Out[... ' sino porque lo único que me faltó fue darte de mamar.» Aureliano escapaba al alba y regresaba a la
madrugada siguiente'

In [... # Limpiamos las frases
frases_limpias = []
for frase in frases:
    #Eliminamos la puntuación y dividimos por espacios
    tokens = frase.translate(str.maketrans(' ',' ',string.punctuation)).split()
    #print(tokens) #para mostrar qué ha hecho hasta aquí
    #Convertimos a minúsculas
    tokens = [word.lower() for word in tokens]
    #print(tokens) #para mostrar qué ha hecho hasta aquí
    if tokens:
        frases_limpias.append(tokens)

In [... # Mostramos los resultados
frases_limpias[500]

Out[... ['las',
'huestes',
'de',
'su',
'padre',
'tenían',
'un',
'aspecto',
'de',
'náufragos',
'sin',
'escapatoria']
```

Entrenamiento del modelo Word2Vec

```
In [...] model = Word2Vec(sentences=frases_limpias,
                         vector_size=500,
                         window=5,
                         min_count=1,
                         workers=4)
```

Explicación:

- sentences: Es la lista de palabras que vamos a vectorizar
- vector_size: Es el tamaño de dimensiones que le daremos al vector
- window: Son la cantidad de palabras por encima y por debajo que le darán contexto
- min_count: La aparición mínima de una palabra para tenerla en cuenta en el entrenamiento
- workers: Cantidad de núcleo de procesador que vamos a invertir en el entrenamiento

Pruebas

```
In [... # Verificamos el vector para alguna palabra
vector = model.wv['padre']
vector

In [... # Mostramos las palabras cercanas
palabras_cercanas = model.wv.most_similar('buendía', topn=10)
palabras_cercanas
# Es probable que la similitud falle por tener tan pocas palabras en el texto
```

```
Out[...]: [('segundo', 0.9980278611183167),  
          ('jósé', 0.9935610890388489),  
          ('buendia', 0.9925903081893921),  
          ('tío', 0.9853704571723938),  
          ('aureliano', 0.9842246770858765),  
          ('promovió', 0.9841890335083008),  
          ('pequeño', 0.9830081462860107),  
          ('participó', 0.9812502264976501),  
          ('informaciones', 0.9800763726234436),  
          ('arcadio', 0.9795325398445129)]
```

Guardar modelo

```
In [...]: model.save('Entrenamiento_Word2Vec/cien_anos_de_soledad.model')
```

Cargar el modelo

```
In [...]: modelo_cargado = Word2Vec.load('Entrenamiento_Word2Vec/cien_anos_de_soledad.model')
```

```
In [...]: # Probamos con el modelo cargado  
palabras_cercanas2 = modelo_cargado.wv.most_similar('buendía', topn=10)  
palabras_cercanas2
```

```
Out[...]: [('segundo', 0.9980278611183167),  
          ('jósé', 0.9935610890388489),  
          ('buendia', 0.9925903081893921),  
          ('tío', 0.9853704571723938),  
          ('aureliano', 0.9842246770858765),  
          ('promovió', 0.9841890335083008),  
          ('pequeño', 0.9830081462860107),  
          ('participó', 0.9812502264976501),  
          ('informaciones', 0.9800763726234436),  
          ('arcadio', 0.9795325398445129)]
```

Guardar Embedido

Existen dos maneras, usando .txt sin binarios, y usando .bin con binarios.

```
In [...]: model.wv.save_word2vec_format('Entrenamiento_Word2Vec/cien_anos_de_soledad_emb.txt', binary=False)  
model.wv.save_word2vec_format('Entrenamiento_Word2Vec/cien_anos_de_soledad_emb.bin', binary=True)
```

Cargar Embedidos

Si se carga el .txt, se usa sin binarios; si se carga el .bin, se usa con binarios

```
In [...]: from gensim.models import KeyedVectors  
embedding_cargado_txt = KeyedVectors.load_word2vec_format(  
    'Entrenamiento_Word2Vec/cien_anos_de_soledad_emb.txt', binary=False)
```

```
In [...]: embedding_cargado_bin = KeyedVectors.load_word2vec_format(  
    'Entrenamiento_Word2Vec/cien_anos_de_soledad_emb.bin', binary=True)
```

```
In [...]: # Probamos  
embedding_cargado_txt
```

```
Out[...]: <gensim.models.keyedvectors.KeyedVectors at 0x1116c4d67b0>
```

```
In [...]: # Probamos  
embedding_cargado_bin
```

```
Out[...]: <gensim.models.keyedvectors.KeyedVectors at 0x1116c43a390>
```

Analogías

```
In [...] def analogics(v1, v2, v3):  
    simil = embedding_cargado_bin.most_similar(positive=[v1,v3],  
                                                negative=[v2]  
                                            )  
    print(f'{v1} es a {v2}, como {simil[0][0]} es a {v3}')
```

```
In [...] analogics('aureliano', 'buendía', 'iguarán')  
aureliano es a buendía, como luz es a iguarán
```

Conclusiones

El texto Cien años de soledad, tiene 823735 caracteres, lo que implica una base de datos más grande para entrenar un modelo, pero sigue siendo pequeña para un modelo adecuado. De cierta forma, el modelo se ajusta en algunos casos puntuales, pero suele mostrar demasiadas stopwords en las similitudes, que tendríamos que manipular para mejorar la predicción de analogías. Veremos con un paquete de textos más grandes y variados, como se generaría la predicción.

Mg. Luis Felipe Bustamante Narváez

Anexo 6

Proyecto 6: Conversión masiva de nombres de archivos

Mg. Luis Felipe Bustamante Narváez

Renombrar

Este ejercicio, permitirá cambiar el nombre de forma masiva a diferentes archivos para ejecutarlos de manera rápida en un análisis de vectorización para NPL, evitando errores por los nombres que traen estos archivos a la hora de descargarlos o de generarlos.

Librerías

In [...]

```
import os
```

Ruta de los archivos a cambiar su nombre

In [...]

```
pathOrigin = 'Entrenamiento_Word2Vec/textos'  
pathFinal = 'Entrenamiento_Word2Vec/textos/textos'
```

Función para renombrar

In [...]

```
def changeNames(originPath, finalPath, prefix):  
    files = os.listdir(originPath)  
    for counter, fileName in enumerate(files):  
        name, extension = os.path.splitext(fileName)  
        newName = f'{prefix}_{counter}{extension}'  
        try:  
            os.rename(os.path.join(originPath, fileName), os.path.join(finalPath, newName))  
            print("Cambio de nombres realizado.")  
        except (err):  
            print("Error al renombrar: ", err)
```

Llamamos la función

In [...]

```
prefix = 'texto'  
changeNames(pathOrigin, pathFinal, prefix)
```

Conclusiones

Este código, permite renombrar listas largas de archivos para una mejor manipulación en entrenamientos de AI.

Mg. Luis Felipe Bustamante Narváez

Anexo 7

Proyecto 7: Creación de Words Embedding Nivel 3

Mg. Luis Felipe Bustamante Narváez

Word2Vec

Es un modelo que se utiliza para aprender representaciones vectoriales de palabras. Estas representaciones pueden capturar muchas propiedades lingüísticas de las palabras, como su significado semántico, gramatical y hasta contextual.

Para este tercer ejemplo, usaremos 100 textos aleatorios, desde 20.000 caracteres hasta más de 1.500.000 caracteres, los cuales suman para el entrenamiento un total de ---- caracteres.

Librerías

```
In [...] !pip install pypdf2
Requirement already satisfied: pypdf2 in c:\users\luis_\anaconda3\envs\notebook\lib\site-packages (3.0.1)

In [...] !pip install tqdm
Requirement already satisfied: tqdm in c:\users\luis_\anaconda3\envs\notebook\lib\site-packages (4.67.1)
Requirement already satisfied: colorama in c:\users\luis_\anaconda3\envs\notebook\lib\site-packages (from tqdm) (0.4.6)

In [...] import string
from gensim.models import Word2Vec
import PyPDF2
import os
from tqdm import tqdm
```

Cargamos el documento

```
In [...] def extraer_texto_desde_pdf(ruta_archivo):
    with open(ruta_archivo, 'rb') as archivo:
        lector = PyPDF2.PdfReader(archivo)
        texto = ''
        for pagina in range(len(lector.pages)):
            texto += lector.pages[pagina].extract_text()
    return texto

In [...] ruta_carpetas = 'Entrenamiento_Word2Vec/textos'
```

Guardamos todos los textos en una lista

```
In [...] todos_los_textos = []
for archivo in tqdm(os.listdir(ruta_carpetas)):
    if archivo.endswith('.pdf'):
```

```

    ruta_completa = os.path.join(ruta_carpetas, archivo)
    try:
        documento = extraer_texto_desde_pdf(ruta_completa)
        todos_los_textos.append(documento)
    except Exception as e:
        print(f'Error al procesar {archivo}: {e}')

```

100%|██████████| 100/100 [12:46<00:00, 7.66s/it]

In [... len(todos_los_textos)

Out[... 100

Procesamiento de datos

El objetivo del procesamiento es convertir el documento en una lista de frases, y cada frase en una lista de palabras, eliminando signos de puntuación y convirtiendo todo a minúsculas.

```

In [... # Dividimos el documento en frases usando La coma como separador
frases_totales = []
caracteres = 0

for documento in todos_los_textos:
    caracteres = caracteres + len(documento)
    frases = documento.split(',')
    frases_totales.extend(frases)

```

In [... # Mostramos el número de caracteres totales
print(f'Número de caracteres: {caracteres}')

Número de caracteres: 240745048

In [... # Mostramos el número de oraciones totales
print(f'El número de frases totales es de: {len(frases_totales)}')

El número de frases totales es de: 1980300

In [... # Mostramos un ejemplo
frases_totales[500]

Out[... '\n\ninfluyendo en el desarrollo de la humanidad en múltiples aspectos. A medida que avanzamos en el\nnconocimiento'

In [... # Mostramos un ejemplo
frases_totales[3000]

Out[... ' nuevas perspectivas emergen'

```

In [... # Limpiamos las frases
frases_limpias = []
for frase in frases_totales:
    #Eliminamos la puntuación y dividimos por espacios
    tokens = frase.translate(str.maketrans(' ', ' ', string.punctuation)).split()
    #print(tokens) #para mostrar qué ha hecho hasta aquí
    #Convertimos a minúsculas
    tokens = [word.lower() for word in tokens]
    #print(tokens) #para mostrar qué ha hecho hasta aquí
    if tokens:
        frases_limpias.append(tokens)

```

In [... # Mostramos Los resultados
frases_limpias[500]

```
Out[... ['influyendo',
    'en',
    'el',
    'desarrollo',
    'de',
    'la',
    'humanidad',
    'en',
    'múltiples',
    'aspectos',
    'a',
    'medida',
    'que',
    'avanzamos',
    'en',
    'el',
    'conocimiento']
```

CPU disponibles en mi PC

En este apartado, observaremos la cantidad de núcleos de procesamiento tiene nuestro computador para el trabajo en NLP. Como este modelo requiere de más gasto computacional, es bueno identificar este dato, para ser eficientes en el entrenamiento, y evitar relentizar el equipo u otros procesos en paralelo.

```
In [... def numero_de_cpus():
    return os.cpu_count()

print(f'Mi equipo tiene {numero_de_cpus()} CPU's')

Mi equipo tiene 8 CPU's
```

Entrenamiento del modelo Word2Vec

```
In [...] model = Word2Vec(sentences=frases_limpias,
                      vector_size=500,
                      window=5,
                      min_count=1,
                      workers=6)
```

Explicación:

- sentences: Es la lista de palabras que vamos a vectorizar
- vector_size: Es el tamaño de dimensiones que le daremos al vector
- window: Son la cantidad de palabras por encima y por debajo que le darán contexto
- min_count: La aparición mínima de una palabra para tenerla en cuenta en el entrenamiento
- workers: Cantidad de núcleo de procesador que vamos a invertir en el entrenamiento

Pruebas

```
In [...] # Verificamos el vector para alguna palabra
vector = model.wv['ciencia']
vector

In [...] # Mostramos las palabras cercanas
palabras_cercanas = model.wv.most_similar('ciencia', topn=10)
```

```
palabras_cercanas
# Es probable que la similitud falle por tener tan pocas palabras en el texto

Out[...]: [('principios', 0.3798693120479584),
          ('tecnología', 0.36144816875457764),
          ('medicina', 0.2695100009441376),
          ('biografía', 0.2573043406009674),
          ('múltiples', 0.2501693665981293),
          ('múltiplesdimensiones', 0.2432234287261963),
          ('economía', 0.24100026488304138),
          ('relevantebiografía', 0.22121661901474),
          ('vidas', 0.20030616223812103),
          ('han', 0.19419683516025543)]
```

```
In [...]: # Mostramos las palabras cercanas
palabras_cercanas = model.wv.most_similar('sistemas', topn=10)
palabras_cercanas
# Es probable que la similitud falle por tener tan pocas palabras en el texto

Out[...]: [('enseñanza', 0.8446722030639648),
          ('educación', 0.4766235649585724),
          ('crisis', 0.4423055946826935),
          ('financieras', 0.4304288625717163),
          ('cultural', 0.37052440643310547),
          ('videojuegos', 0.35479822754859924),
          ('actualidad', 0.29080677032470703),
          ('gastronomía', 0.25270408391952515),
          ('fundamentales', 0.2261238843202591),
          ('aportes', 0.2239352911710739)]
```

Guardar modelo

```
In [...]: model.save('Entrenamiento_Word2Vec/100textos.model')
```

Cargar el modelo

```
In [...]: modelo_cargado = Word2Vec.load('Entrenamiento_Word2Vec/100textos.model')

In [...]: # Probamos con el modelo cargado
palabras_cercanas2 = modelo_cargado.wv.most_similar('importancia', topn=10)
palabras_cercanas2

Out[...]: [('emocional', 0.6839113831520081),
          ('industria', 0.29906395077705383),
          ('cultura', 0.2832416296005249),
          ('másimportantes', 0.24401310086250305),
          ('expertos', 0.2300969511270523),
          ('energía', 0.22232283651828766),
          ('géneros', 0.22123292088508606),
          ('importantes', 0.20623861253261566),
          ('nuclear', 0.19350115954875946),
          ('descubrimientos', 0.18687954545021057)]
```

Guardar Embedido

Existen dos maneras, usando .txt sin binarios, y usando .bin con binarios.

```
In [...] model.wv.save_word2vec_format('Entrenamiento_Word2Vec/100textos_emb.txt', binary=False)
model.wv.save_word2vec_format('Entrenamiento_Word2Vec/100textos_emb.bin', binary=True)
```

Cargar Embedidos

Si se carga el .txt, se usa sin binarios; si se carga el .bin, se usa con binarios

```
In [...] from gensim.models import KeyedVectors
embedding_cargado_txt = KeyedVectors.load_word2vec_format(
    'Entrenamiento_Word2Vec/100textos_emb.txt', binary=False)
```

```
In [...] embedding_cargado_bin = KeyedVectors.load_word2vec_format(
    'Entrenamiento_Word2Vec/100textos_emb.bin', binary=True)
```

```
In [...] # Probamos
embedding_cargado_txt
```

```
Out[...] <gensim.models.keyedvectors.KeyedVectors at 0x1d45dbc0a70>
```

```
In [...] # Probamos
embedding_cargado_bin
```

```
Out[...] <gensim.models.keyedvectors.KeyedVectors at 0x1d444aa0710>
```

Analogías

```
In [...] def analogics(v1, v2, v3):
    simil = embedding_cargado_bin.most_similar(positive=[v1,v3],
                                                negative=[v2])
    print(f'{v1} es a {v2}, como {simil[0][0]} es a {v3}')
```

```
In [...] analogics('científico', 'ciencia', 'cultura')
científico es a ciencia, como pensamiento es a cultura
```

```
In [...] analogics('científico', 'ciencia', 'nuclear')
científico es a ciencia, como energía es a nuclear
```

```
In [...] analogics('científico', 'ciencia', 'imperio')
científico es a ciencia, como romano es a imperio
```

Conclusiones

Utilizando 100 textos con temáticas aleatorias, se puede observar que las predicciones en las analogías son mucho más reales; chatGPT3 utilizó en su primer entrenamiento 570G en textos, libros y artículos, nuestro entrenamiento utilizó tan solo 82.8M y aún así, encontramos mucha coherencia a la hora de probar la similitud. ¿Qué pasaría si usáramos por lo menos 1.000.000 de textos?

Anexo 8

Proyecto 8: Clasificador de texto

Mg. Luis Felipe Bustamante Narváez

En este ejercicio realizaremos un clasificador de texto basado en la forma escritural, sintáctica y semántica de dos escritores latinoamericanos, por un lado al argentino Jorge Luis Borges, y por otro frente al uruguayo Mario Benedetti.



Jorge Luis Borges



Mario Benedetti

1. Temas y Filosofía

Borges: Su poesía es filosófica, abstracta y llena de referencias literarias, mitológicas y metafísicas. Le interesaban temas como el tiempo, el infinito, el destino, la identidad y la memoria. Su tono es intelectual y a veces enigmático.

Benedetti: Escribe de manera más directa y accesible. Sus temas son el amor, la vida cotidiana, la lucha social, el exilio y la esperanza. Su tono es cálido, humano y cercano al lector.

2. Lenguaje y Estilo

Borges: Usa un lenguaje elegante, erudito y con muchas metáforas complejas. Su poesía es reflexiva, con estructuras clásicas y a veces con formas fijas como sonetos.

Benedetti: Usa un lenguaje sencillo, directo y coloquial. Sus poemas parecen conversaciones o pensamientos escritos sin mucha ornamentación.

3. Estructura y Ritmo

Borges: Tiende a usar estructuras más tradicionales con rima y métrica cuidadas, aunque también experimenta con versos libres.

Benedetti: Prefiere el verso libre y la naturalidad del habla cotidiana, sin preocuparse demasiado por la métrica.

Librerías

```
In [...] pip install tensorflow
In [...] pip install colorama
In [...] import numpy as np
import matplotlib.pyplot as plt
import string
from sklearn.model_selection import train_test_split
import os
from tqdm import tqdm
import PyPDF2
from IPython.display import display, HTML
from tensorflow.keras.preprocessing.sequence import pad_sequences
from colorama import Fore, Back, Style
```

Cargamos los documentos

```
In [...] def extraer_texto_desde_pdf(ruta_archivo):
    with open(ruta_archivo, 'rb') as archivo:
        lector = PyPDF2.PdfReader(archivo)
```

```

        texto = ''
        for pagina in range(len(lector.pages)):
            texto += lector.pages[pagina].extract_text()
        return texto

```

In [...] ruta_carpetas = 'textos'

Guardamos los textos en una lista

```

In [...] todos_los_textos = []
for archivo in tqdm(os.listdir(ruta_carpetas),
                     bar_format=f'{Back.WHITE}{Fore.GREEN}{{l_bar}} {{bar}}{Style.RESET_ALL}'):
    if archivo.endswith('.pdf'):
        ruta_completa = os.path.join(ruta_carpetas, archivo)
        try:
            documento = extraer_texto_desde_pdf(ruta_completa)
            todos_los_textos.append(documento)
        except Exception as e:
            print(f'Error al procesar {archivo}: {e}')

```

100% |██████████|

In [...] todos_los_textos[0]

Procesamiento de los datos

Vamos a separar los textos por etiquetas, enumerando los textos de Borges con la etiqueta 0 y los de Benedetti con la etiqueta 1

```

In [...] # Eliminamos espacios al inicio y al final para evitar problemas con el pdf
for texto in todos_los_textos:
    archivo = texto.strip()
archivo

```

```

In [...] # Creamos las listas vacías
textos = []
etiquetas = []

```

```

In [...] # Mostramos línea por línea los textos de cada escritor
#count = 0 #contador de linea para pruebas
for etiqueta, texto in enumerate(todos_los_textos):
    print(f'\n--- Texto {etiqueta} ---\n')
    for linea in texto.split('\n'):
        #count += 1 #contador de líneas para prueba
        print(linea)
        # Convertimos a minúsculas
        linea = linea.rstrip().lower()
        print(linea)
        # Eliminamos signos de puntuación
        if linea:
            linea = linea.translate(str.maketrans(' ', ' ', string.punctuation))
            print(linea)
            # Agregamos el texto limpio y le asignamos su respectiva etiqueta
            textos.append(linea)
            etiquetas.append(etiqueta)

```

```

In [...] # Mostramos las listas
textos

```

Entrenamiento

X representa la lista de los textos, y Y, representa la lista de las etiquetas, quien sería nuestra variable a predecir.

```

In [...] X_train, X_test, Y_train, Y_test = train_test_split(textos, etiquetas, train_size=0.9, random_state=42)

```

```

In [...] # Mostramos en forma de tupla el tamaño de cada muestra
len(Y_train), len(Y_test)

```

Out[... (2222, 247)

```
In [... # Probamos las muestras de entrenamiento  
X_train[0], Y_train[0]
```

```
Out[... ('o acaso no la mira', 1)
```

```
In [... # Probamos las muestras de prueba  
X_test[0], Y_test[0]
```

```
Out[... ('a inventar la verdad', 1)
```

Representación de palabras desconocidas

<unk>

Es una convención utilizada a menudo en **NPL** para representar palabras desconocidas o fuera del vocabulario. Por ejemplo, si una palabra no se encontró en la muestra de entrenamiento, pero aparece en la muestra de prueba, será desconocida, y se requiere agregarle un índice que diferencie a esta palabra.

```
In [... indice = 1  
indice_palabras = {'<unk>': 0}
```

Construcción del diccionario de codificación de palabras a índice

```
In [... for texto in X_train:  
    # Separamos cada palabra  
    tokens = texto.split()  
    #print(tokens) # Probamos como se ven los tokens  
    for token in tokens:  
        # Buscamos si la palabra no está en el índice para luego agregarla sin repetir  
        if token not in indice_palabras:  
            indice_palabras[token] = indice  
            indice += 1
```

```
In [... # Mostramos el índice de palabras - palabras únicas  
indice_palabras
```

```
In [... # tamaño de palabras únicas  
indice_palabras
```

Conversión del índice de palabras de String a enteros

Como el entrenamiento no se debe hacer con palabras, creamos una muestra convertida a su valor específico en enteros

```
In [... # Listas para enteros  
X_train_int = []  
X_test_int = []  
# Banderas para ejecutarse una sola vez  
X_int_train_hecho = False  
X_int_test_hecho = False
```

```
In [... # Conversión de los datos de entrenamiento  
if not X_int_train_hecho:  
    for texto in X_train:  
        # dividimos de nuevo en palabras  
        tokens = texto.split()  
        # Por cada palabra encontrada la cambia por su valor numérico de la clave del diccionario  
        linea_entero = [ indice_palabras[token] for token in tokens ]  
        #print(linea_entero)  
        # Agregamos el nuevo valor a la lista de entrenamiento  
        X_train_int.append(linea_entero)  
    X_int_train_hecho = True  
    print("Conversión de entrenamiento ejecutada con éxito.")  
else:  
    print("La conversión de entrenamiento ya se había ejecutado previamente.")
```

Conversión de entrenamiento ejecutada con éxito.

```
In [... # Mostramos la conversión de entrenamiento
X_train_int

In [... # Conversión de los datos de prueba -- Como puede haber desconocidos, debemos hacer esto:
if not X_int_test_hecho:
    for texto in X_test:
        tokens = texto.split()
        linea_entero = [indice_palabras.get(token, 0) for token in tokens] #trae el token o 0
        #print(linea_entero)
        X_test_int.append(linea_entero)
    X_int_test_hecho = True
    print("Conversión de prueba ejecutada con éxito.")
else:
    print("La conversión de prueba ya se había ejecutado previamente.")

Conversión de prueba ejecutada con éxito.
```

```
In [... # Mostramos la conversión de prueba
len(X_test_int)
```

Out[... 247

Matriz de Transición

Como se indicó en la teoría de los procesos de **Markov**, se requiere construir una matriz de transición y los estados iniciales para cada escritor:

1. Creamos un vector **V** con el tamaño total del **indice_palabras**
2. Creamos la matriz **A0** para las palabras de **Borges**
3. Creamos el vector de probabilidad inicial **pi0** para las palabras de **Borges**
4. Creamos la matriz **A1** para las palabras de **Benedetti**
5. Creamos el vector de probabilidad inicial **pi1** para las palabras de **Benedetti**

```
In [... V = len(indice_palabras)
# Creamos las matrices y vectores con 1 para poder hacer el suavizado
A0 = np.ones((V, V))
pi0 = np.ones(V)
A1 = np.ones((V, V))
pi1 = np.ones(V)
#Motramos, por ejemplo
pi0
```

Out[... array([1., 1., 1., ..., 1., 1., 1.])

Función de conteo de palabras

```
In [... def compute_counts(texto_as_int, A, pi):
    #Recorremos los tokens
    for tokens in texto_as_int:
        #Creamos el posible último elemento como referencia
        last_index = None
        #Recorremos cada elemento de cada línea
        for index in tokens:
            #Nos ubicamos en la primera secuencia
            if last_index is None:
                # Agregamos el valor inicial
                pi[index] += 1
            else:
                # Agregamos los valores a la matriz
                A[last_index, index] += 1
            # Asignamos el valor actual al last_index
            last_index = index
```

```
In [... # Llamamos la función
#Para Borges
compute_counts([t for t, y in zip(X_train_int, Y_train) if y == 0], A0, pi0)
#Para Benedetti
compute_counts([t for t, y in zip(X_train_int, Y_train) if y == 1], A1, pi1)
```

```
In [...] # Probamos
A1
```

```
Out[... array([[1., 1., 1., ..., 1., 1., 1.],
   [1., 1., 2., ..., 1., 1., 1.],
   [1., 1., 1., ..., 1., 1., 1.],
   ...,
   [1., 1., 1., ..., 1., 1., 1.],
   [1., 1., 1., ..., 1., 1., 1.],
   [1., 1., 1., ..., 1., 1., 1.]])
```

Explicación

```
pi0 = array([ 1., 10., 1., ..., 1., 1., 1.])
```

```
pi1 = array([ 1., 14., 1., ..., 1., 1., 1.])
```

```
A0 = array([[1., 1., 1., ..., 1., 1., 1.], [1., 1., 1., ..., 1., 1., 1.], [1., 1., 1., ..., 1., 1., 1.], ..., [1., 1., 1., ..., 1., 1., 1.], [1., 1., 1., ..., 1., 1., 1.], [1., 1., 1., ..., 1., 1., 1.]]))
```

```
A1 = array([[1., 1., 1., ..., 1., 1., 1.], [1., 1., 2., ..., 1., 1., 1.], [1., 1., 1., ..., 1., 1., 1.], ..., [1., 1., 1., ..., 1., 1., 1.], [1., 1., 1., ..., 1., 1., 1.], [1., 1., 1., ..., 1., 1., 1.]]))
```

En el vector inicial **pi0**, la primera posición corresponde a los unk, la segunda posición corresponde a la palabra **o**, y nos está indicando que en los textos de Borges aparece iniciando la línea 10 veces. Si observamos los textos de Benedetti, **pi1**, nos indica que aparece 14 veces comenzando la línea. De esta manera podemos proceder a encontrar la probabilidad.

Con respecto a las matrices de transición, podemos observar en la matriz de Benedetti, **A1**, que en la segunda fila, hubo una transición de la palabra actual a la siguiente, valores que nos indicarán el comportamiento natural de los textos.

Distribución de Probabilidad

Normalizamos

Para observar las probabilidades, se requiere normalizar los vectores y matrices generados en el conteo, para que su valor oscile entre 0 y 1, como debe ser.

Esta es una manera empírica de demostrar las fórmulas mencionadas en la teoría

```
In [...] # Conservamos los datos originales A0, A1, pi0 y pi1, creando las variables nomralizadas
# Para esto vamos a guardar los datos originales
A0_norm = A0.copy()
pi0_norm = pi0.copy()
A1_norm = A1.copy()
pi1_norm = pi1.copy()
# Bandera de normalizado
normalize = False
```

```
In [...] # Borges
if not normalize:
    # Borges
    A0_norm /= A0_norm.sum(axis=1, keepdims = True)
    pi0_norm /= pi0_norm.sum()
    # Benedetti
    A1_norm /= A1_norm.sum(axis=1, keepdims = True)
    pi1_norm /= pi1_norm.sum()
    print("Normalización ejecutada con éxito.")
    normalize = True
else:
    print("Las variables ya fueron normalizadas previamente.")
```

Normalización ejecutada con éxito.

```
In [...] # Probamos
pi0_norm
```

```
Out[... array([0.00023111, 0.00231107, 0.00023111, ..., 0.00023111, 0.00023111,
0.00023111])
```

```
In [...] A0_norm
```

```
Out[...] array([[0.00029472, 0.00029472, 0.00029472, ..., 0.00029472, 0.00029472,
0.00029472],
[0.00029155, 0.00029155, 0.00029155, ..., 0.00029155, 0.00029155,
0.00029155],
[0.00029455, 0.00029455, 0.00029455, ..., 0.00029455, 0.00029455,
0.00029455],
...,
[0.00029472, 0.00029472, 0.00029472, ..., 0.00029472, 0.00029472,
0.00029472],
[0.00029472, 0.00029472, 0.00029472, ..., 0.00029472, 0.00029472,
0.00029472],
[0.00029464, 0.00029464, 0.00029464, ..., 0.00029464, 0.00029464,
0.00029464]])
```

Explicación

```
pi0_norm = array([0.00023111, 0.00231107, 0.00023111, ..., 0.00023111, 0.00023111, 0.00023111])
```

```
A0_norm = array([[0.00029472, 0.00029472, 0.00029472, ..., 0.00029472, 0.00029472, 0.00029472], [0.00029155, 0.00029155, ..., 0.00029155, 0.00029155, 0.00029155], [0.00029455, 0.00029455, 0.00029455, ..., 0.00029455, 0.00029455, 0.00029455], ..., [0.00029472, 0.00029472, 0.00029472, ..., 0.00029472, 0.00029472, 0.00029472], [0.00029472, 0.00029472, 0.00029472, ..., 0.00029472, 0.00029472, 0.00029472], [0.00029464, 0.00029464, 0.00029464, ..., 0.00029464, 0.00029464, 0.00029464], 0.00029464]))
```

En el vector inicial **pi0**, observemos que no aparece ningún valor en cero, lo que indica que el método de suavizar funcionó perfectamente, al igual que en la matriz **A0**, lo que permite una **distribución de probabilidad**.

Espacio logarítmico

Como vimos en la teoría, estas probabilidades pueden tener un desbordamiento por debajo, ya que se aproximan a cero, entonces, para evitar errores computacionales, usaremos el espacio logarítmico.

```
In [...] #Borges
log_A0_norm = np.log(A0_norm)
log_pi0_norm = np.log(pi0_norm)
#Benedetti
log_A1_norm = np.log(A1_norm)
log_pil_norm = np.log(pil_norm)
```

```
In [...] # Probamos
log_pi0_norm
```

```
Out[...] array([-8.37262974, -6.07004465, -8.37262974, ..., -8.37262974,
-8.37262974, -8.37262974])
```

```
In [...] # Probamos
log_A0_norm
```

```
Out[...] array([[ -8.12946976, -8.12946976, -8.12946976, ..., -8.12946976,
-8.12946976, -8.12946976],
[-8.14031554, -8.14031554, -8.14031554, ..., -8.14031554,
-8.14031554, -8.14031554],
[-8.13005904, -8.13005904, -8.13005904, ..., -8.13005904,
-8.13005904, -8.13005904],
...,
[-8.12946976, -8.12946976, -8.12946976, ..., -8.12946976,
-8.12946976, -8.12946976],
[-8.12946976, -8.12946976, -8.12946976, ..., -8.12946976,
-8.12946976, -8.12946976],
[-8.12976445, -8.12976445, -8.12976445, ..., -8.12976445,
-8.12976445, -8.12976445]])
```

Pre-análisis

Vamos a revisar diferentes elementos que nos permitan entender mejor lo que desarrollamos

```
In [...]
# Conteo de etiquetas de clase 0 (Borges) en Y_train
count_Y_0 = sum(y == 0 for y in Y_train)
# Conteo de etiquetas de clase 1 (Benedetti) en Y_train
count_Y_1 = sum(y == 1 for y in Y_train)
# Cantidad total de ejemplos de entrenamiento
total = len(Y_train)
# Probabilidad a priori de la clase 0
p0 = count_Y_0 / total
# Probabilidad a priori de la clase 1
p1 = count_Y_1 / total
# Logaritmo de la clase a priori 0
log_p0 = np.log(p0)
# Logaritmo de la clase a priori 0
log_p1 = np.log(p1)

display(HTML(f'''  

Se encontró {count_Y_0} etiquetas de clase 0, <b style='color:fuchsia;'>Borges</b>,<br>
Se encontró {count_Y_1} etiquetas de clase 1, <b style='color:skyblue;'>Benedetti</b>,<br>
para un total de <b style='color:red;'>{total}</b> ejemplos de entrenamiento.  

<hr>
Las probabilidades a priori serían las siguientes:<br>
<table style="border: 1px solid black; border-collapse: collapse;">
    <tr>
        <td style="border: 1px solid black; padding: 5px;">Borges</td>
        <td style="border: 1px solid black; padding: 5px;">{p0}</td>
    </tr>
    <tr>
        <td style="border: 1px solid black; padding: 5px;">Benedetti</td>
        <td style="border: 1px solid black; padding: 5px;">{p1}</td>
    </tr>
</table>
<hr>
Como usamos el espacio logarítmico, estas serían las probabilidades reales de encontrar un texto de la clase 0 o 1:  

<table style="border: 1px solid black; border-collapse: collapse;">
    <tr>
        <td style="border: 1px solid black; padding: 5px;">Borges</td>
        <td style="border: 1px solid black; padding: 5px;">{log_p0}</td>
    </tr>
    <tr>
        <td style="border: 1px solid black; padding: 5px;">Benedetti</td>
        <td style="border: 1px solid black; padding: 5px;">{log_p1}</td>
    </tr>
</table>
<hr>
'''))
```

Se encontró 934 etiquetas de clase 0, **Borges**,

Se encontró 1288 etiquetas de clase 1, **Benedetti**,

para un total de **2222** ejemplos de entrenamiento.

Las probabilidades a priori serían las siguientes:

Borges	0.42034203420342037
Benedetti	0.5796579657965797

Como usamos el espacio logarítmico, estas serían las probabilidades reales de encontrar un texto de la clase 0 o 1:

Borges	-0.8666865319707326
Benedetti	-0.5453170635352763

Construcción del Clasificador

```
In [...]
# Creamos una clase
class Classifier:
    # Constructor
    def __init__(self, log_As, log_pis, log_apriors):
        self.log_As = log_As
        self.log_pis = log_pis
        self.log_apriors = log_apriors
        # número de clases
        self.k = len(log_apriors)

    # Método de verosimilitud
    def _compute_log_likelihood(self, input_, class_):
        log_A = self.log_As[class_]
        log_pi = self.log_pis[class_]
        #Repetimos lo hecho en el ejemplos de creación de la matriz
        last_index = None
        log_prob = 0
        #Recorremos la entrada del usuario
        for index in input_:
            if last_index is None:
                #Primer token en la secuencia
                log_prob += log_pi[index]
            else:
                #Calculamos la probabilidad de transición de la palabra anterior a la actual
                log_prob += log_A[last_index, index]
                #Actualizamos el index para la próxima iteración
                last_index = index
        return log_prob

    # Función de predicción
    def predict(self, inputs):
        predictions = np.zeros(len(inputs))
        for i, input_ in enumerate(inputs):
            # Calcula los Logaritmos de las probabilidades posteriores para cada clase
            posteriors = [self._compute_log_likelihood(input_, c) + self.log_apriors[c] \
                         for c in range(self.k)]
            #Elije la clase de mayor probabilidad posterior como la predicción
            pred = np.argmax(posteriors)
            predictions[i] = pred
        return predictions
```

Explicación

1 Constructor (`__init__`) ✨ ¿Qué parámetros recibe?

`log_As`: Matrices de probabilidades de transición entre palabras en logaritmo. `log_pis`: Probabilidades iniciales de cada palabra en logaritmo. `log_apriors`: Probabilidades previas (prior) de cada clase en logaritmo. `self.k`: Número total de clases.

💡 ¿Por qué usa logaritmos? ✅ Evita problemas de underflow cuando se multiplican muchas probabilidades pequeñas. ✅ Convierte productos en sumas, lo que hace más fácil la optimización.

2 Método `_compute_log_likelihood` (Cálculo de Verosimilitud)

💡 ¿Qué hace?

Calcula la log-verosimilitud de una secuencia (`input_`) dada una clase (`class_`). Usa la probabilidad inicial de la primera palabra (`log_pi[index]`). Luego, suma las probabilidades de transición entre palabras (`log_A[last_index, index]`). Retorna `log_prob`, que indica qué tan probable es la secuencia dada la clase.

3 Método `predict` (Clasificación)

💡 ¿Qué hace?

Inicializa `predictions` con ceros (un array para almacenar las predicciones).

Para cada entrada en `inputs`: Calcula las log-verosimilitudes para cada clase. Suma la probabilidad previa (prior) `log_apriors[c]` de cada clase. Elije la clase con mayor probabilidad posterior usando `np.argmax()`. Devuelve `predictions`, que contiene las clases predichas.

Objeto de la clase Clasifier

```
In [... # Creamos un objeto de la clase Clasifier para llamar los métodos del clasificador
clf = Classifier([log_A0_norm, log_A1_norm], [log_pi0_norm, log_pi1_norm], [log_p0, log_p1])
```

Explicación

La clase Classifier, recibe 3 parámetros en su constructor, es decir 3 atributos. En su orden estos atributos son:

- 1 [log_A0_norm, log_A1_norm] que serán los argumentos de log_As
- 2 [log_pi0_norm, log_pi1_norm] que serán los argumentos de log_pis
- 3 [log_p0, log_p1] que serán los argumentos de log_apriors

Es decir,

- 1 Las matrices de transición normalizadas
- 2 Los vectores con los valores iniciales o estados iniciales
- 3 Las probabilidades de cada clase utilizando el espacio logarítmico

Predicción

```
In [... # Llamamos al método predict (Datos de entrenamiento: Aprox 1.0)
P_train = clf.predict(X_train_int)
# Mostramos la predicción con la muestra de entrenamiento
print(f'Accuraci Train: {np.mean(P_train == Y_train)}')]
```

Accuraci Train: 0.9932493249324933

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

Out[... 247

```
In [... # Llamamos al método predict (Datos de prueba)
P_test = clf.predict(X_test_int)
# Mostramos la predicción con la muestra de prueba
print(f'Accuraci Test: {np.mean(P_test == Y_test)}')]
```

Accuraci Test: 0.7530364372469636

Probamos con textos nuevos

📖 Textos en los estilos de Borges y Benedetti

Le solicitamos a ChatGPT4 que nos generara textos con escritura similar a cada uno de nuestros poetas, y esto fue lo que nos entregó.

Estilo Borges

*En el vasto archivo del tiempo,
donde las sombras del ayer se funden
con las visiones inciertas del mañana,
el hombre camina por un laberinto
de palabras y azares.
Sabe, aunque lo olvida a menudo,
que su destino está tejido con hilos invisibles,
urdidos por manos que jamás verá.*

*Alguna tarde,
quizá en la penumbra de una biblioteca
o en la geometría secreta de un sueño,
descubrirá que su vida no ha sido sino el eco de otras vidas,*

*el reflejo de una historia que ya ha sido escrita
en un idioma remoto y perfecto.*

Estilo Benedetti

*La ciudad despierta con su ritmo de siempre,
entre murmullos de bocacalles
y pasos apurados que no saben bien a dónde van.*

*En un café cualquiera,
un hombre revuelve su taza con la mirada perdida,
tal vez recordando un amor que ya no está,
tal vez soñando con la ternura que aún no llega.*

*Afuera, la vida sigue su curso,
con su cuota justa de olvidos y de esperanzas,
con sus rutinas que a veces duelen y a veces salvan.*

*Porque, al final de cuentas,
lo importante no es cuánto nos golpea el tiempo,
sino con quién elegimos compartirlo.*

Funciones para poner a prueba el nuevo texto

```
In [...]
def preprocesar_texto(texto):
    texto = texto.lower().translate(str.maketrans(' ', ' ', string.punctuation))
    tokens = texto.split()
    return tokens

def convertir_a_indices(tokens, indice_palabras):
    return [indice_palabras.get(token, 0) for token in tokens] # 0 para <unk>

def calcular_probabilidad(tokens_indices, log_pi, log_A):
    if not tokens_indices:
        return -np.inf # Evitar errores con texto vacío
    log_prob = log_pi[tokens_indices[0]] # Probabilidad inicial
    for i in range(len(tokens_indices) - 1):
        log_prob += log_A[tokens_indices[i], tokens_indices[i + 1]] # Probabilidad de transición
    return log_prob

def clasificar_texto(texto, log_pi0_norm, log_A0_norm, log_pi1_norm, log_A1_norm, indice_palabras):
    tokens = preprocesar_texto(texto)
    tokens_indices = convertir_a_indices(tokens, indice_palabras)

    log_prob_borges = calcular_probabilidad(tokens_indices, log_pi0_norm, log_A0_norm)
    log_prob_benedetti = calcular_probabilidad(tokens_indices, log_pi1_norm, log_A1_norm)

    #Conversión de LOG-PROBABILIDADES a PROBABILIDADES
    max_log_prob = max(log_prob_borges, log_prob_benedetti) # Para evitar underflow numérico
    prob_borges = np.exp(log_prob_borges - max_log_prob)
    prob_benedetti = np.exp(log_prob_benedetti - max_log_prob)
    total_prob = prob_borges + prob_benedetti
    prob_borges /= total_prob # Normalizamos
    prob_benedetti /= total_prob

    print(f"Probabilidad Borges: {prob_borges:.4f}")
    print(f"Probabilidad Benedetti: {prob_benedetti:.4f}")

    return "Borges" if prob_borges > prob_benedetti else "Benedetti"
```

Explicación

Este código implementa un clasificador de texto que determina si un texto nuevo se asemeja más a un escrito de Borges o de Benedetti. Lo hace utilizando un modelo basado en cadenas de Markov, donde se calculan probabilidades de transición entre palabras.

1 preprocesar_texto(texto)

Objetivo: Convierte el texto en minúsculas, elimina signos de puntuación y lo divide en palabras (tokens).

Proceso:

1. texto.lower() → Convierte todo el texto a minúsculas.
2. .translate(str.maketrans(", ", string.punctuation)) → Elimina la puntuación.
3. .split() → Divide el texto en palabras, generando una lista de tokens.

2 convertir_a_indices(tokens, indice_palabras)

Objetivo: Convierte cada palabra (token) en un índice numérico basado en un diccionario indice_palabras.

Proceso:

1. Usa indice_palabras.get(token, 0) para obtener el índice de cada palabra en el diccionario.
2. Si la palabra no está en el diccionario, se asigna 0 (se usa para , palabras desconocidas).

3 calcular_probabilidad(tokens_indices, log_pi, log_A)

Objetivo: Calcula la log-probabilidad de una secuencia de palabras en base a un modelo de cadenas de Markov.

Proceso:

1. Si tokens_indices está vacío, retorna -np.inf (evita errores).
2. Toma la log-probabilidad inicial de la primera palabra: log_pi[tokens_indices[0]].
3. Para cada par de palabras consecutivas en la secuencia, suma la log-probabilidad de transición log_A.

4 clasificar_texto(...)

Objetivo: Clasifica un texto como "Borges" o "Benedetti" en función de su probabilidad de generación en cada modelo.

Proceso:

1. Preprocesa el texto → Obtiene los tokens.
2. Convierte los tokens a índices → Transforma palabras en números.
3. Calcula log-probabilidades para ambos modelos (Borges y Benedetti).
4. Convierte log-probabilidades a probabilidades reales para interpretación:
 - Se resta el máximo log-probabilidad para evitar problemas numéricos (underflow).
 - Se aplica np.exp(log_prob - max_log_prob) para convertir log a probabilidad.
 - Se normaliza dividiendo entre la suma total.
5. Imprime las probabilidades y retorna la clasificación.

Ingreso del nuevo texto - Similar a Borges

```
In [...] texto_prueba = """
En la vastedad de la memoria,
donde los ecos de los días transcurridos
se entrelazan con los espejismos del porvenir,
el hombre camina por un laberinto
de palabras y azares.

Sabe, aunque lo olvida a menudo,
que su destino está tejido con hilos invisibles,
urdidos por manos que jamás verá.

Alguna tarde,
quizá en la penumbra de una biblioteca
o en la geometría secreta de un sueño,
descubrirá que su vida no ha sido sino el eco de otras vidas,
el reflejo de una historia que ya ha sido escrita
en un idioma remoto y perfecto.
"""

resultado = clasificar_texto(texto_prueba, log_pi0_norm, log_A0_norm, log_pi1_norm, log_A1_norm, indice_palabras)
print(f"Clasificación: {resultado}")
```

```
Probabilidad Borges: 1.0000
Probabilidad Benedetti: 0.0000
Clasificación: Borges
```

Ingreso del nuevo texto - Similar a Benedetti

```
In [...]
texto_prueba = """
La ciudad despierta con su ritmo de siempre,
entre murmullos de bocacalles
y pasos apurados que no saben bien a dónde van.

En un café cualquiera,
un hombre revuelve su taza con la mirada perdida,
tal vez recordando un amor que ya no está,
tal vez soñando con la ternura que aún no llega.

Afuera, la vida sigue su curso,
con su cuota justa de olvidos y de esperanzas,
con sus rutinas que a veces duelen y a veces salvan.

Porque, al final de cuentas,
lo importante no es cuánto nos golpea el tiempo,
sino con quién elegimos compartirlo.
"""

resultado = clasificar_texto(texto_prueba, log_pi0_norm, log_A0_norm, log_pi1_norm, log_A1_norm, indice_palabras)
print(f"Clasificación: {resultado}")

Probabilidad Borges: 0.0028
Probabilidad Benedetti: 0.9972
Clasificación: Benedetti
```

Guardar como .py

```
In [...]
# !jupyter nbconvert --to script Proyecto8.ipynb
```

Conclusiones

Se puede observar como los datos de prueba, permiten clasificar los textos con un 75% de presición, de tal manera que al poner textos de estos poetas, fácilmente podrá indicar quien lo escribió.

Mg. Luis Felipe Bustamante Narváez

Anexo 9

Proyecto 9: Generador de texto

Mg. Luis Felipe Bustamante Narváez

En este ejercicio realizaremos un generador de texto, basado en el archivo pdf trabajado en el clasificador [mario-benedetti.pdf](#). Este archivo, nos permitirá entrenar un modelo de **Markov** de **Segundo Orden**.

Tengamos en cuenta los siguientes detalles de la manera de escribir del autor:



Mario Benedetti

Mario Benedetti (1920-2009), fue un escritor, poeta y periodista uruguayo, considerado una de las figuras más importantes de la literatura latinoamericana. Nació el 14 de septiembre de

1920 en Paso de los Toros, Uruguay. Su obra abarcó poesía, narrativa, ensayo y teatro, con un estilo sencillo y cercano que exploraba el amor, la memoria, el exilio y la lucha social.

Durante la dictadura en Uruguay (1973-1985), se exilió en varios países como Argentina, Cuba y España, lo que marcó profundamente su escritura. Entre sus libros más conocidos se encuentran La tregua (1960), Gracias por el fuego (1965) y El amor, las mujeres y la vida (1995). Benedetti falleció el 17 de mayo de 2009 en Montevideo, dejando un legado literario que sigue conmoviendo a lectores de todas las generaciones.

📘 1. Temas y Filosofía

Escribe de manera directa y accesible. Sus temas son el amor, la vida cotidiana, la lucha social, el exilio y la esperanza. Su tono es cálido, humano y cercano al lector.

✍ 2. Lenguaje y Estilo

Usa un lenguaje sencillo, directo y coloquial. Sus poemas parecen conversaciones o pensamientos escritos sin mucha ornamentación.

⌚ 3. Estructura y Ritmo

Prefiere el verso libre y la naturalidad del habla cotidiana, sin preocuparse demasiado por la métrica.

Librerías

```
In [...]
import numpy as np
import string
import PyPDF2
import itertools
```

Convertimos el pdf en .txt

En esta oportunidad, como ya conocemos el manejo de pdf's, vamos a usar un conversor a .txt, para agilizar la manipulación de los datos.

```
In [...] def pdf_to_text(pdf_path, txt_path):
    with open(pdf_path, 'rb') as pdf_file:
        reader = PyPDF2.PdfReader(pdf_file)
        text = ''
        for page in reader.pages:
            text += page.extract_text() + '\n'
    with open(txt_path, 'w', encoding='utf-8') as txt_file:
        txt_file.write(text)

    print(f'Texto extraído y guardado en "{txt_path}"')
```

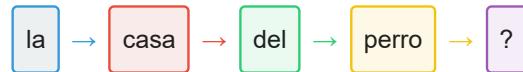
```
In [...] # Ruta pdf
pdf = 'textos/mario_benedetti.pdf'
txt = 'textos/mario_benedetti.txt'
pdf_to_text(pdf, txt)
```

Texto extraído y guardado en "textos/mario_benedetti.txt"

Diccionarios

Diccionario inicial

Supongamos que tenemos la siguiente frase:



Necesitamos obtener la primera palabra de cada frase, y las veces que dicha palabra aparece dentro del texto como primera palabra.

Por ejemplo, en el diagrama anterior, la palabra **la** aparece como primera palabra de la frase **la casa del perro** si suponemos que es la única frase de nuestro texto, entonces dicha palabra se repite **1** vez.

Luego, el diccionario inicial quedaría así:

```
pa_inicial = { 'la': 1 }
```

```
In [...] pa_inicial = {}
```

Diccionario de Markov Primero Orden

Supongamos que tenemos las siguientes frases:



Necesitamos obtener la primera palabra de cada frase, y las veces que dicha palabra aparece dentro del texto como primera palabra, como vimos anteriormente, además necesitamos saber cuales son las secuencias de primer orden, para componer nuestro diccionario.

Luego, el diccionario inicial y el diccionario de primer orden quedarían así:

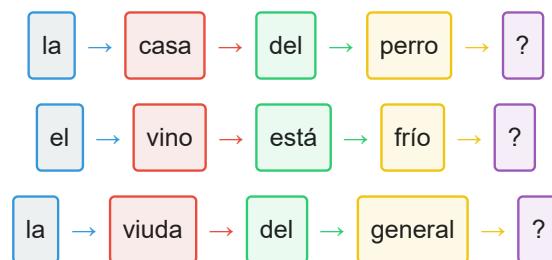
```
pa_inicial = { 'la': 2, 'el': 1 }
```

```
primer_orden = { 'la': ['casa', 'viuda'],
                  'casa': ['del'],
                  'del': ['perro', 'general'],
                  'perro': ['END'],
                  'el': ['vino'],
                  'vino': ['frío'],
                  'frío': ['END'],
                  'viuda': ['del'],
                  'general': ['END']
                }
```

```
In [... primer_orden = {}]
```

Diccionario de Markov Segundo Orden

Supongamos para el ejemplo anterior:



Necesitamos el conjunto de palabras que anteceden a una palabra cualquiera, para ello, usamos un diccionario donde la clave sean las dos palabras inmediatamente anteriores en el tiempo.

Luego, el diccionario de segundo orden quedaría así:

```
segundo_orden = { ('la', 'casa'): ['del'],
                   ('casa', 'del'): ['perro'],
                   ('el', 'vino'): ['está'],
                   ('vino', 'está'): ['frío'],
                   ('la', 'viuda'): ['del'],
                   ('viuda', 'del'): ['general']
                 }
```

Notemos, que las claves de cada diccionario, son conjuntos de palabras.

```
In [...] segundo_orden = {}

In [...] # Función para remover puntuación y poner en minúsculas
def remove_punct_lower(txt):
    txt = txt.translate(str.maketrans(' ', ' ', string.punctuation))
    txt = txt.lower()
    return txt

In [...] # Función para añadir valores al diccionario
def add_dict(dicc, key, value):
    if key not in dicc:
        dicc[key] = []
    dicc[key].append(value)
```

Cargamos el archivo

```
In [...] # Reiniciar diccionarios
pa_inicial = {}
primer_orden = {}
segundo_orden = {}

In [...] # Si necesitamos reiniciar este código para efectos prácticos, basta con reiniciar los dict
with open('textos/mario_benedetti.txt', 'r', encoding='utf-8') as archivo:
    for linea in archivo:
        #print(linea) #comentar
        #Llamamos a la función
        tokens = remove_punct_lower(linea).split()
        #print(tokens) #comentar
        T = len(tokens)
        #print(f'Tamaño de la Línea: {T}') #Comentar
        #Recorremos los elementos de la fila
        for i in range(T):
            token = tokens[i]
            if i == 0:
                pa_inicial[token] = pa_inicial.get(token, 0.) + 1
                #print(f'Palabra inicial: {token}') #comentar
            else:
                t_1 = tokens[i-1]
                #últimas 2 palabras de cada frase
                if i == T-1:
                    add_dict(segundo_orden, (t_1, token), 'END')
                    #palabras con una sola palabra previa
                if i == 1:
                    add_dict(primer_orden, t_1, token)
                else:
                    t_2 = tokens[i-2]
                    add_dict(segundo_orden, (t_2, t_1), token)

In [...] dict(itertools.islice(pa_inicial.items(), 5))

Out[...] {'poemas': 3.0, 'mario': 1.0, 'la': 26.0, 'una': 16.0, 'genera': 1.0}

In [...] dict(itertools.islice(segundo_orden.items(), 5))
```

```
Out[... {('poemas', 'varios'): ['END'],
         ('mario', 'benedetti'): ['END'],
         ('buena', 'tiniebla'): ['END'],
         ('la', 'buena'): ['tiniebla', 'fe', 'suerte'],
         ('una', 'mujer'): ['desnuda', 'desnuda', 'desnuda', 'querida', 'dice']}
```

```
In [...] dict(itertools.islice(primer_orden.items(), 3))
```

```
Out[... {'poemas': ['varios', 'a', 'al'],
        'mario': ['benedetti'],
        'la': ['buena',
               'madriguera',
               'lluvia',
               'vida',
               'patria',
               'política',
               'claridad',
               'primavera',
               'misma',
               'luz',
               'válida',
               'madre',
               'incógnita',
               'pareja',
               'pareja',
               'despareja',
               'gente',
               'cosa',
               'querida',
               'culpa',
               'culpa',
               'ebriedad',
               'política',
               'muerte',
               'generosidad',
               'soledad']}
```

Explicación

Analicemos los resultados de los cinco primeros elementos de cada uno de los diccionarios creados a partir del texto.

1. pa_inicial:

```
{'poemas': 3.0, 'mario': 1.0, 'la': 26.0, 'una': 16.0, 'genera': 1.0}
```

Indica, por ejemplo que, la palabra **la** aparece como palabra inicial **16.0**.

2. primer_orden:

```
{
    'poemas': [
        'varios',
        'a',
        'al'
    ],
    'mario': [
        'benedetti'
```

```

],
'la': [
    'buena',
    'madriguera',
    'lluvia',
    'vida',
    .
    .
    .
    'política',
    'muerte',
    'generosidad',
    'soledad'
]
}

```

Indica, por ejemplo que, después de la palabra 'la' aparecen las siguientes palabras: 'buena', 'madriguera', 'lluvia', etc..

3. segundo_orden:

```

{
('poemas', 'varios'): ['END'],
('mario', 'benedetti'): ['END'],
('buena', 'tiniebla'): ['END'],
('la', 'buena'): [
    'tiniebla',
    'fe',
    'suerte'
],
('una', 'mujer'): [
    'desnuda',
    'desnuda',
    'desnuda',
    'querida',
    'dice'
]
}

```

Indica, por ejemplo que, después de las palabras ('la', 'buena') aparecen las siguientes palabras: 'tiniebla', 'fe' y 'suerte'.

Probabilidades

Normalización

Palabras iniciales

```
In [...] # Calculamos el total de apariciones de todas las palabras iniciales
initial_total = sum(pa_inicial.values())
initial_total
```

```
Out[...] 1474.0
```

```
In [...] # Creamos el diccionario inicial de probabilidad
pa_inicial_prob = pa_inicial.copy()
```

```

for key, value in pa_inicial.items():
    pa_inicial_prob[key] = value / inicial_total

In [...] dict(itertools.islice(pa_inicial_prob.items(), 3))

Out[...] {'poemas': 0.0020352781546811396,
          'mario': 0.0006784260515603799,
          'la': 0.017639077340569877}

```

Función de conversión a probabilidad

```

In [...] # Función para convertir las listas de primer orden en diccionarios
# 'poemas': ['varios', 'a', 'al'],
def list_to_pdicc(listas):
    # Creamos un diccionario vacío
    d = {}
    # Obtenemos la Longitud de la Lista de elementos
    n = len(listas)

    # Ciclo para contar la ocurrencia de cada elemento en la lista
    for lista in listas:
        d[lista] = d.get(lista, 0.) + 1

    # Ciclo para convertir los conteos en probabilidades relativas
    for key, value in d.items():
        d[key] = value / n

    # Retornamos el diccionario de probabilidades
    return d

```

Primer Orden

```

In [...] # Llamamos la función de probabilidad de primer orden
primer_orden_prob = primer_orden.copy()
for t_1, t in primer_orden.items():
    primer_orden_prob[t_1] = list_to_pdicc(t)

In [...] dict(itertools.islice(primer_orden_prob.items(), 2))

Out[...] {'poemas': {'varios': 0.3333333333333333,
                      'a': 0.3333333333333333,
                      'al': 0.3333333333333333},
          'mario': {'benedetti': 1.0}}

```

Segundo Orden

```

In [...] # Llamamos la función de probabilidad de segundo orden
segundo_orden_prob = segundo_orden.copy()
for s, t in segundo_orden.items():
    segundo_orden_prob[s] = list_to_pdicc(t)

In [...] dict(itertools.islice(segundo_orden_prob.items(), 5))

Out[...] {('poemas', 'varios'): {'END': 1.0},
          ('mario', 'benedetti'): {'END': 1.0},
          ('buena', 'tiniebla'): {'END': 1.0},
          ('la', 'buena'): {'tiniebla': 0.3333333333333333,
                           'fe': 0.3333333333333333,
                           'suerte': 0.3333333333333333},
          ('una', 'mujer'): {'desnuda': 0.6, 'querida': 0.2, 'dice': 0.2}}

```

Función de prueba

Con esta función, realizaremos las pruebas necesarias para evaluar el comportamiento de las probabilidades de aparición de una palabra.

La función recibe dos parámetros, **diccionario** e **imprimir**; el primer parámetro recibe el diccionario de trabajo a evaluar, y el segundo recibe un booleano, para saber si mostramos o no los pasos de la ejecución.

```
In [... def palabra_ejemplo(d, imp):
      # Generamos un número aleatorio en el rango [0, 1]
      p0 = np.random.random()
      if imp:
          print(f"p0:\t\t{p0}")

      # Inicializamos una variable para realizar la suma acumulativa de probabilidades
      cumulative = 0
      if imp:
          print(f"prob. acum.: \t{cumulative}")

      # Ciclo que recorre cada clave (k) y su probabilidad (p) en el diccionario (d)
      for k, p in d.items():
          cumulative += p
          if imp:
              print(f"Prob:\t\t{p}\titem:\t{k}")
              print(f"prob. acum.: \t{cumulative}")

      # Comprobamos si el número aleatorio es menor que la acumulación de probabilidades
      if p0 < cumulative:
          # Si la condición se cumple, devuelve la clave (k) seleccionada
          respuesta = f"La palabra siguiente debería ser '{k}'"
          return respuesta if imp else k
```

```
In [... # Verificamos que nos traería una palabra en particular
      print(primer_orden_prob['poemas'])

      {'varios': 0.3333333333333333, 'a': 0.3333333333333333, 'al': 0.3333333333333333}
```

```
In [... # Llamamos la función
      palabra_ejemplo(primer_orden_prob['poemas'], True)
```

```
p0:           0.36180865610514357
prob. acum.:  0
Prob:         0.3333333333333333      item:    'varios'
prob. acum.:  0.3333333333333333
Prob:         0.3333333333333333      item:    'a'
prob. acum.:  0.6666666666666666
```

```
Out[... "La palabra siguiente debería ser 'a'"
```

Generador

Función del generador de texto

```
In [... def generador(tamaño):
      for i in range(tamaño):
          oracion = []
          #Palabra inicial
          pal_0 = palabra_ejemplo(pa_inicial_prob, False)
```

```

oracion.append(pal_0)
#segunda palabra
pal_1 = palabra_ejemplo(primer_orden_prob[pal_0], False)
oracion.append(pal_1)

# Segundo orden hasta el final
while True:
    pal_2 = palabra_ejemplo(segundo_orden_prob[(pal_0, pal_1)], False)
    if pal_2 == 'END':
        break
    oracion.append(pal_2)
    pal_0 = pal_1
    pal_1 = pal_2
texto = ' '.join(oracion)
print(texto)

```

Explicación

- Esta función recibe un parámetro tamaño, que representa la cantidad de oraciones que se generarán:

def generador (tamaño):

- Se inicia un bucle que se ejecutará **tamaño** veces, es decir, generará **tamaño** oraciones.

for i in range (tamaño):

- Se crea una lista vacía llamada oracion, donde se almacenarán las palabras generadas.

oracion = []

- Se elige la primera palabra de la oración con la función **palabra_ejemplo(pa_inicial_prob, False)**. **pa_inicial_prob** es un diccionario que contiene probabilidades de palabras las iniciales. La palabra seleccionada se agrega a oracion.

pal_0 = palabra_ejemplo(pa_inicial_prob, False)
oracion.append (pal_0)

- Se elige la segunda palabra, dependiendo de la primera (**pal_0**). **primer_orden_prob** es el diccionario que mapea palabras iniciales a posibles segundas palabras con sus probabilidades. **palabra_ejemplo(primer_orden_prob[pal_0], False)** selecciona la segunda palabra basada en **pal_0**. Se agrega **pal_1** a **oracion**.

pal_1 = palabra_ejemplo(primer_orden_prob [pal_0], False)
oracion.append (pal_1)

- Se entra en un bucle while True para generar palabras de manera recurrente. **segundo_orden_prob** es un diccionario que usa tuplas (**pal_0, pal_1**) como clave y devuelve las probabilidades de la siguiente palabra. Se

selecciona la siguiente palabra (**pal_2**) en función de las dos palabras anteriores.

```
while True:
```

```
    pal_2 = palabra_ejemplo(segundo_orden_prob [ (pal_0, pal_1) ], False )
```

- Si **pal_2** es '**END**', se termina la generación de palabras y se sale del bucle.

```
if pal_2 == 'END':  
    break
```

- Si **pal_2** no es '**END**', se agrega a la lista oracion. Luego, se actualizan las variables: **pal_0 toma el valor de pal_1** y **pal_1 toma el valor de pal_2**. Esto asegura que en la siguiente iteración se utilicen las dos palabras más recientes para predecir la siguiente.

```
oracion.append (pal_2)  
pal_0 = pal_1  
pal_1 = pal_2
```

- La lista oracion se convierte en una cadena de texto separada por espacios con '**'join(oracion)**'. Se imprime la oración generada.

```
texto = ' '.join (oracion)  
print (texto)
```

```
In [... # Probamos la creación de un poema, de acuerdo con La manera de escritura de Mario Benedetti  
generador(5)
```

```
mi nombre con su hermano el insociable  
en esas noches en que fui un viejo cargado de recelos  
mientras los grandes temas  
la muerte  
pero no viceversa por algo en el aire que absorbieron noche a noche
```

Conclusiones

Aquí construimos un generador de texto basado en modelos de Markov de segundo orden. usando probabilidades de aparición de palabras para construir oraciones de manera secuencial hasta encontrar una palabra de terminación ('END').

Mg. Luis Felipe Bustamante Narváez

Anexo 10

Proyecto 10: Spinning de Texto

Mg. Luis Felipe Bustamante Narváez

Librerías

```
In [...]
import numpy as np
import pandas as pd
import nltk
from nltk import word_tokenize
from nltk.tokenize.treebank import TreebankWordDetokenizer
import asyncio
from tqdm import tqdm
from colorama import Fore, Back, Style
import os
from itertools import islice
from IPython.display import display, Markdown
```

```
In [...]
# Descargamos el conjunto de datos del tokenizador en español
nltk.download('punkt')

[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\luis_\AppData\Roaming\nltk_data...
[nltk_data]     Package punkt is already up-to-date!
```

Out[...]: True

Cargamos los datos

```
In [...]
# Es común que los archivos vengan codificados con ISO
path = 'data/data_larazon_publico_v2.csv'
path_utf = 'data/new_data.csv'
try:
    df = pd.read_csv(path, encoding='utf-8')
    print('Encoding utf-8')
except Exception:
    print('Encoding ISO-8859-1 a utf-8')
    df_iso = pd.read_csv(path, encoding='ISO-8859-1')
    df_iso.to_csv(path_utf, encoding='utf-8', index=False)
#await asyncio.sleep(3) #Espera 3 seg para abrir el nuevo archivo en espera de ser guardado
df = pd.read_csv(path_utf, encoding='utf-8')
```

Encoding utf-8

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

	Unnamed: 0	indi	cuerpo	titular
0	0	0	dos semanas después de su puesta de largo y pr...	el submarino s-80 ya flota
1	1	1	este viernes, el presidente del gobierno, pedr...	calviño y calvo alaban (sin darse cuenta) la g...
2	2	2	el ministro del interior, fernando grande-marl...	el geo de la policía tendrá una nueva sede en ...
3	3	3	son días muy duros para la familia de olivia y...	la madre de las niñas "sobran las palabras par...
4	4	4	sólo quedan 10 presos de eta por recibir los b...	sólo quedan 10 presos de eta por recibir el be...
...
58419	58420	18419	la comisión europea inició este un procedimien...	bruselas abre un expediente a españa por no de...
58420	58421	18420	el pleno de la asamblea de madrid ha aprobado ...	aprobado el proyecto de ley para que las mujer...
58421	58422	18421	la comisión de investigación parlamentaria del...	la comisión del alvia arranca escuchando a la ...
58422	58423	18422	erc y pdecat han calificado este jueves de "in...	erc y pdecat piden explicaciones a interior po...
58423	58424	18423	la junta de portavoces del congreso ha acordad...	el congreso aplaza la primera sesión de contro...

58424 rows × 4 columns

In [... df.head()

	Unnamed: 0	indi	cuerpo	titular
0	0	0	dos semanas después de su puesta de largo y pr...	el submarino s-80 ya flota
1	1	1	este viernes, el presidente del gobierno, pedr...	calviño y calvo alaban (sin darse cuenta) la g...
2	2	2	el ministro del interior, fernando grande-marl...	el geo de la policía tendrá una nueva sede en ...
3	3	3	son días muy duros para la familia de olivia y...	la madre de las niñas "sobran las palabras par...
4	4	4	sólo quedan 10 presos de eta por recibir los b...	sólo quedan 10 presos de eta por recibir el be...

Creamos la Serie con las noticias

In [... # Tomamos solamente la columna abstract para crear una serie
textos = df['cuerpo']

In [... textos.head()

Out[... 0 dos semanas después de su puesta de largo y pr...
1 este viernes, el presidente del gobierno, pedr...
2 el ministro del interior, fernando grande-marl...
3 son días muy duros para la familia de olivia y...
4 sólo quedan 10 presos de eta por recibir los b...
Name: cuerpo, dtype: object

In [... #muestra de las noticias (solo las primeras 500 palabras)
textos[0][:500]

Out[... 'dos semanas después de su puesta de largo y presentación en sociedad, el primer submarino s-80 para la armada, el s-81 "isaac peral", ha entrado hoy en el agua tras una delicada y larga maniobra que se ha retrasado varios días por las condiciones meteorológicas. de esta forma, tras completar su construcción 17 años después de que arrancara el programa, navantia ha cumplido otro importante hito.españa.submarino s-80 tras 17 años y 3.900 millones, el "isaac peral" ya está aquí.españa.el comandante '

Probabilidades

Matriz de conteo

In [... # Creamos el diccionario de probabilidad
key: (w(t-1), w(t+1)), value: {w(t): count(w(t))}
probs = {}

```
In [...]
# Separador
#variable de formato de la barra de progreso
bar_format_ = (f'{Back.WHITE}{Fore.GREEN}{{l_bar}}{{bar}}{Style.RESET_ALL}' +
               f'{Fore.CYAN}{{n_fmt}}/{{total_fmt}}' +
               f'[{{elapsed}}<{{remaining}}]{Style.RESET_ALL}' +
)
for doc in tqdm(textos, bar_format=bar_format_, desc='Creando matriz: '):
    #Separamos cada noticia por puntos
    lineas = doc.split('.')
    for linea in lineas:
        #Tokenizamos cada linea
        tokens = word_tokenize(linea, language='spanish')
        #Mostramos los tokens
        #print(tokens) #Este proceso tarda bastante, se hace a modo de prueba
        #Condicionamos las palabras finales
        if len(tokens) >= 2:
            for i in range(len(tokens) - 2):
                t_0 = tokens[i] #palabra anterior
                t_1 = tokens[i+1] #palabra actual
                t_2 = tokens[i+2] #palabra siguiente
                #Creamos la clave del diccionario
                key = (t_0, t_2)
                #preguntamos si la clave no está en el diccionario
                if key not in probs:
                    #asinamos una clave vacía
                    probs[key] = {}
                #preguntamos si la palabra actual no es una clave
                if t_1 not in probs[key]:
                    #asignamos valor inicial de 1 al diccionario de valores de las probs
                    probs[key][t_1] = 1
                else:
                    #sumamos el valor de aparición de la palabra actual
                    probs[key][t_1] += 1

    #mostramos las líneas a modo de prueba
    #lineas
```

100% |██████████| 58424/58424 [06:03<00:00]

```
In [...]
# Mostramos el diccionario probs, pero solo una parte para hacer corto el proceso
dict(islice(probs.items(),1))
```

```
Out[...]
{('dos', 'después'): {'semanas': 95,
                      'años': 283,
                      'días': 296,
                      'meses': 208,
                      'horas': 35,
                      'siglos': 4,
                      'minutos': 4,
                      'décadas': 16,
                      'elecciones': 2,
                      'día': 3,
                      'jornadas': 2,
                      'legislaturas': 1,
                      'domingos': 1,
                      'negocios': 1,
                      'pasiones': 1,
                      'decenios': 1,
                      'iniciativas': 1,
                      'dispositivos': 1}}
```

```
In [... len(probs)
```

```
Out[... 4875993
```

Normalización

```
In [... # Creamos una copia del diccionario para mantener los datos  
d_probs = probs.copy()  
# Recorremos las claves y los valores del diccionario probs  
for key, d in tqdm(d_probs.items(), bar_format=bar_format_, desc='Normalizando: '):  
    # sumamos los valores de repetición de cada una de las palabras  
    total = sum(d.values())  
    # Recorremos la clave y el valor del diccionario de los valores creados  
    for k, v in d.items():  
        d[k] = v / total
```

```
Normalizando: 100%|██████████| 4875993/4875993 [00:21<00:00]
```

```
In [... # Mostramos el diccionario d_probs, pero solo una parte para hacer corto el proceso  
dict(islice(d_probs.items(),1))
```

```
Out[... {('dos', 'después'): {'semanas': 0.09947643979057591,  
'años': 0.2963350785340314,  
'días': 0.3099476439790576,  
'meses': 0.21780104712041884,  
'horas': 0.03664921465968586,  
'siglos': 0.004188481675392671,  
'minutos': 0.004188481675392671,  
'décadas': 0.016753926701570682,  
'elecciones': 0.0020942408376963353,  
'día': 0.0031413612565445027,  
'jornadas': 0.0020942408376963353,  
'legislaturas': 0.0010471204188481676,  
'domingos': 0.0010471204188481676,  
'negocios': 0.0010471204188481676,  
'pasiones': 0.0010471204188481676,  
'decenios': 0.0010471204188481676,  
'iniciativas': 0.0010471204188481676,  
'dispositivos': 0.0010471204188481676}]}
```

```
In [... len(d_probs)
```

```
Out[... 4875993
```

Ejemplo de Detokenización

Permite volver a unir los tokens en frases, por ejemplo:

```
In [... detokenizar = TreebankWordDetokenizer()  
ejemplo = 'Bootcamp de Inteligencia Artificial'  
print(f'Frase original: {ejemplo}')  
token_ejemplo = word_tokenize(ejemplo, language='spanish')  
print(f'Frase tokenizada: {token_ejemplo}')  
detoken_ejemplo = detokenizar.detokenize(token_ejemplo)  
print(f'Frase Detokenizada: {detoken_ejemplo}')
```

```
Frase original: Bootcamp de Inteligencia Artificial  
Frase tokenizada: ['Bootcamp', 'de', 'Inteligencia', 'Artificial']  
Frase Detokenizada: Bootcamp de Inteligencia Artificial
```

Spinner

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

```
# Función de prueba para una palabra random
def sample_word(d):
    p0 = np.random.random()
    cumulative = 0
    for key, p in d.items():
        cumulative += p
        if p0 < cumulative:
            return key
```

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

```
# Función spinner para una Línea
# CADA COMENTARIO DONDE ESTÁ EL RETURN ES UN EJEMPLO PARA IR ANALIZANDO EL CÓDIGO
def spin_line(linea, imp):
    tokens = word_tokenize(linea, language='spanish')
    i = 0
    salida = [tokens[0]]
    #return salida #ejemplo de ejecución --- comentar
    if len(tokens) >= 2:
        while i < (len(tokens) - 2):
            t_0 = tokens[i] #palabra anterior
            t_1 = tokens[i+1] #palabra actual
            t_2 = tokens[i+2] #palabra siguiente
            #Creamos la clave
            key = (t_0, t_2)
            #Creamos el diccionario de distribución
            p_dist = d_probs[key]
            #i = 1100000 #Para desbordar el while ----- comentar
            #return p_dist #ejemplo de ejecución ----- comentar
            #Cuando el diccionario tenga más de una palabra y un spinning del x%
            if len(p_dist) > 1 and np.random.random() < 0.3:
                #selecciona una palabra al azar de la función de prueba de palabras
                middle = sample_word(p_dist)
                #i = 1100000 #Para desbordar el while ----- comentar
                #return middle #ejemplo de ejecución ----- comentar

                #Validamos si deseamos mostrar la palabra de cambio automáticamente
                # Si imp es True, muestra el texto cambiado
                # Si imp es False, muestra la palabra actual y el cambio que sugiere
                if imp:
                    #agregamos la palabra nueva en la posición t_1
                    salida.append(middle)
                    #agregamos la palabra t_2, que va al final
                    salida.append(t_2)
                    #movemos el cursor 2 posici. para que no haga 2 spin en 2 pal. seguidas
                    i += 2
                else:
                    #agregamos a la salida la palabra t_1, es decir la que queremos cambiar
                    salida.append(t_1)
                    #agregamos, para visualizar, la palabra por la que nos va a cambiar
                    salida.append('<' + middle + '>')
                    #agregamos la palabra t_2, que va al final
                    salida.append(t_2)
                #movemos el cursor dos posici. para que no haga dos spin en 2 pal seguidas
                i += 2
            #en caso que el diccionario sea <= 1 o que el random no entre al spinner
            else:
                #agregamos la palabra siguiente y ubicamos el cursor en la siguiente palabra
```

```

        salida.append(t_1)
        i += 1
    # si ya estamos en la última palabra a poner a prueba
    if i == len(tokens) - 2:
        #agregamos La última palabra al diccionario
        salida.append(tokens[-1])
    # retornamos la salida detokenizada ya que es una lista ESTE NO SE COMENTA, ES EL FIN
    detoken = detokenizar.detokenize(salida)
    return detoken

```

In [...]

```

# Función spinner para recorrer el documento
def spin_document(doc, imp):
    lineas = doc.split('.')
    output = []
    for linea in lineas:
        if linea:
            new_line = spin_line(linea, imp)
        else:
            new_line = linea
        output.append(new_line)
    #corregimos el posible error de tener cadenas en None
    try:
        return '\n'.join(output)
    except Exception:
        return '\n'.join(filter(None, output))

```

In [...]

```

#Código para pruebas de creación
#spin_document('dos años después cómo están')
#spin_line('dos años después cómo están')

```

Texto (noticia) de prueba para el modelo

In [...]

```

#Recordemos qué tenía nuestro df textos
textos.head()

```

Out[...]

0	dos semanas después de su puesta de largo y pr...
1	este viernes, el presidente del gobierno, pedr...
2	el ministro del interior, fernando grande-marl...
3	son días muy duros para la familia de olivia y...
4	sólo quedan 10 presos de eta por recibir los b...

Name: cuerpo, dtype: object

In [...]

```

#seleccionamos un índice cualquiera de alguna noticia del df textos
i = np.random.choice(textos.shape[0])
display(Markdown('---'))
display(Markdown(f'**Índice seleccionado:** {i}'))
display(Markdown('---'))
#tomamos el texto que se encuentra en dicho índice
doc = textos.iloc[i]
#Recortamos el texto, solo para mostrarlo; no se altera el texto inicial
doc_recortado = doc.split() #separamos el texto en palabras
doc_recortado = ' '.join(doc_recortado[:100])
display(Markdown(f'**Texto seleccionado:**'))
print(f'{doc_recortado}...')
display(Markdown('---'))

```

#Generamos el Spinning Article - Text

```

imp = True
new_doc = spin_document(doc, imp)

#Recortamos el nuevo texto, solo para mostrarlo; no se altera el texto generado por el spin
new_doc_recortado = new_doc.split() #separamos el texto en palabras
new_doc_recortado = ' '.join(new_doc_recortado[:100])
display(Markdown(f'**Texto Spinning:**\n\n'))
print(f'{new_doc_recortado}...')
display(Markdown('---'))

```

Índice seleccionado: 35576

Texto seleccionado:

un centenar de taxistas se han concentrado a las 8,00 horas de este viernes en la entrada del cementerio de la almudena de madrid y, pasada esta hora, tenían bloqueada la zona. esta acción se produce en el marco de las protestas que está protagonizando el colectivo desde el pasado lunes por la regulación del sector. se trata de la primera acción del día, que irá acompañada de otras como el comienzo de la huelga de hambre que van a iniciar 16 compañeros a partir de las 10,00 horas de hoy en los alrededores de ifema, centro de operaciones de...

Texto Spinning:

un centenar de taxistas se han concentrado a las 2,00 horas de este campo en la entrada del cementerio de la almudena de presentarlo y la pasada esta hora, tenían bloqueada la zona esta cifra se produce en el toque de las protestas que está protagonizando el colectivo según el próximo lunes por una regulación del sector se trate de la primera acción del día después que irá acompañada de otras sin el comienzo de la historia de hambre que maltratan a iniciar 16 compañeros a cambio de las 10,00 horas de hoy mantener los sistemas de ifema, instructor de...

Errores de tipo NoneType - Análisis

Cuando existe un valor None en el output de la función spin_document, no se puede definir el nuevo texto sugerido. Para solucionar, basta con filtrar el output antes de hacer el join.

```
'\n'.join(filter(None, output))
```

Conclusiones

El Article Spinning, permite realizar cambios de palabras con el fin de brindar otra opción a un texto ya construido y cambiarle sus palabras de modo que conserve la idea contextual, pero con otro estilo de escritura. El uso de N-Grams a través de las cadenas de Markov, permiten utilizar las probabilidades de ocurrencia de una palabra cuando ésta se encuentra en medio de dos palabras previamente entrenadas. Aunque el modelo es bueno, se requiere de un filtro de fuentes más preciso de un tema en específico, pero este es un sencillo ejemplo que nos deja el desafío de usar Spinning Text dentro de NPL.

Anexo 11

Proyecto 11: Detector de Spam

Mg. Luis Felipe Bustamante Narváez

Este proyecto se desarrollará utilizando una base de datos sintética creada a través de ChatGPT con base en datasets extraídos en inglés de diferentes repositorios. Esto debido a, no tener muchas fuentes de gran tamaño de correos electrónicos en español, y no poder hacer uso de ciertas bases de datos por motivos de seguridad. Además, se utilizará una base de datos tomada de kaggle, donde se muestran correos en inglés, para validar que sin importar el idioma, se cumple con el objetivo.

Para desarrollar este ejercicio, usaremos Naive Bayes, con el fin de clasificar si un correo electrónico es o no, spam.

Librerías

```
In [...] pip install seaborn -q
Note: you may need to restart the kernel to use updated packages.

In [...] pip install wordcloud -q
Note: you may need to restart the kernel to use updated packages.

In [...] import numpy as np
import pandas as pd
import seaborn as sn
import matplotlib.pyplot as plt
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_auc_score, f1_score, confusion_matrix
from sklearn.naive_bayes import MultinomialNB
from wordcloud import WordCloud #Mostrar gráfico de palabras
```

Cargamos los datos

```
In [...] opcion = int(input('Ingrese 1 para los correos en español y 2 para los correos en inglés:'))

if opcion == 1:
    path = 'data/spam.csv'
    df = pd.read_csv(path, encoding='utf-8')
elif opcion == 2:
    path = 'data/spam_or_not_spam.csv'
    df = pd.read_csv(path, encoding='ISO-8859-1')
    df.rename(columns={'email':'contenido', 'label':'spam'}, inplace=True)
    #encontramos una cadena NAN df[df.isna().any(axis=1)]
    df['contenido'] = df['contenido'].fillna('') #La reemplazamos por cadena vacía
else:
    print('Opción no válida')

In [...] df
```

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

	contenido	spam
0	date wed NUMBER aug NUMBER NUMBER NUMBER NUMBER NUMB...	0
1	martin a posted tassos papadopoulos the greek ...	0
2	man threatens explosion in moscow thursday aug...	0
3	klez the virus that won t die already the most...	0
4	in adding cream to spaghetti carbonara which ...	0
...
2995	abc s good morning america ranks it the NUMBE...	1
2996	hyperlink hyperlink hyperlink let mortgage le...	1
2997	thank you for shopping with us gifts for all ...	1
2998	the famous ebay marketing e course learn to s...	1
2999	hello this is chinese traditional à¤à»í NUM...	1

3000 rows × 2 columns

```
In [...]: df['contenido'][0][:500]
```

```
Out[...]: 'date wed NUMBER aug NUMBER NUMBER NUMBER NUMBER from chris garrigues cwg dated NUMB  
ER NUMBERfaNUMBERd deepeddy com message id NUMBER NUMBER tmda deepeddy vircio com i can t re  
produce this error for me it is very repeatable like every time without fail this is the deb  
ug log of the pick happening NUMBER NUMBER NUMBER pick_it exec pick inbox list lbrace lbrace  
subject ftp rbrace rbrace NUMBER NUMBER sequence mercury NUMBER NUMBER NUMBER exec pick inbo  
x list lbrace lbrace subject ftp rbrace '
```

```
In [...]: df['spam'][0]
```

```
Out[...]: 0
```

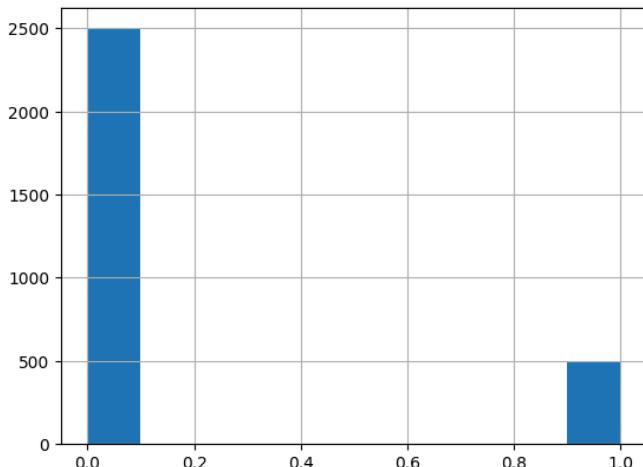
```
In [...]: # Agrupamos para obtener el total de datos (0-> ham, 1->spam)  
grouped = df.groupby('spam').count()  
grouped
```

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

spam	contenido
0	2500
1	500

```
In [...]: # Gráfico de histograma de la población  
df['spam'].hist()
```

```
Out[...]: <Axes: >
```



```
In [...]: # Convertimos la columna spam en un arreglo  
Y = df['spam'].to_numpy()
```

```
In [...] Y  
Out[...] array([0, 0, 0, ..., 1, 1, 1], dtype=int64)
```

Procesamiento de Datos

Entrenamiento

```
In [...] # Dividimos los datos en conjuntos de entrenamiento y de prueba  
df_train, df_test, Y_train, Y_test = train_test_split(df['contenido'], Y, test_size=0.2)
```

```
In [...] df_train
```

```
Out[...] 244      zimbabwe has dropped objections to accepting ...  
1157      on wed aug NUMBER NUMBER at NUMBER NUMBER ulis...  
2418      url URL date NUMBER NUMBER NUMBERtNUMBER NUMBE...  
171       david asked my wife noticed something odd the ...  
1458      on wed NUMBER sep NUMBER stephane lentz wrote ...  
          ...  
294       URL an investigation has been launched after ...  
1644      your mail and gives you only the non spam to ...  
2648      unlimited web conferencing subscribe to the w...  
2599      free adult lifetime membership limited time o...  
948       from valdis kletnieks URL date mon NUMBER aug...  
Name: contenido, Length: 2400, dtype: object
```

```
In [...] df_test
```

```
Out[...] 1585     i m taking all my razored mail today and calli...  
1495      URL jm URL changed what removed added status ...  
855       original message from gary lawrence murphy ga...  
1171      help i had gpg working i updated from version ...  
2227      url URL date NUMBER NUMBER NUMBERtNUMBER NUMBE...  
          ...  
675       help me out here you around barely but don t ...  
90        hi dermot if have a look at one of the dists l...  
197       hey i has just been given an old toshiba csNUM...  
1506      unable to find user matt_relay sbcglobal net p...  
354       on wed NUMBER NUMBER NUMBER at NUMBER NUMBER g...  
Name: contenido, Length: 600, dtype: object
```

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

```
Out[...] 2400
```

Vectorizamos

```
In [...] vectores = CountVectorizer(decode_error='ignore')  
X_train = vectores.fit_transform(df_train)  
X_test = vectores.transform(df_test)
```

```
In [...] X_train
```

```
Out[...] <2400x31260 sparse matrix of type '<class 'numpy.int64'>'  
with 284220 stored elements in Compressed Sparse Row format>
```

```
In [...] X_test
```

```
Out[...] <600x31260 sparse matrix of type '<class 'numpy.int64'>'  
with 61327 stored elements in Compressed Sparse Row format>
```

Modelo

```
In [... model = MultinomialNB()  
model.fit(X_train, Y_train)
```

```
Out[... ▾ MultinomialNB ⓘ ⓘ  
MultinomialNB()
```

```
In [... # Probamos el modelo con los datos originales  
train_accuracy = model.score(X_train, Y_train)  
test_accuracy = model.score(X_test, Y_test)
```

```
In [... # Mostramos la puntuación  
print(f'El accuracy de entrenamiento es de {train_accuracy}')  
print(f'El accuracy de prueba es de {test_accuracy}')  
  
El accuracy de entrenamiento es de 0.9954166666666666  
El accuracy de prueba es de 0.9983333333333333
```

```
In [... # Probamos la predicción del modelo  
P_train = model.predict(X_train)  
P_test = model.predict(X_test)
```

```
In [... # Mostramos el ajuste del modelo predicho con f1  
print(f'Train F1: {f1_score(Y_train, P_train)}')  
print(f'Test F1: {f1_score(Y_test, P_test)}')  
  
Train F1: 0.9864029666254636  
Test F1: 0.994535519125683
```

Matriz de Confusión

```
In [... conf_matrix_train = confusion_matrix(Y_train, P_train)  
conf_matrix_train
```

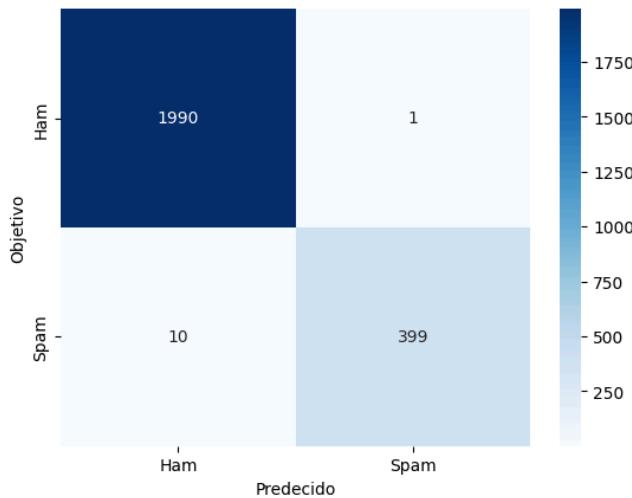
```
Out[... array([[1990,    1],  
              [ 10,  399]], dtype=int64)
```

```
In [... conf_matrix_test = confusion_matrix(Y_test, P_test)  
conf_matrix_test
```

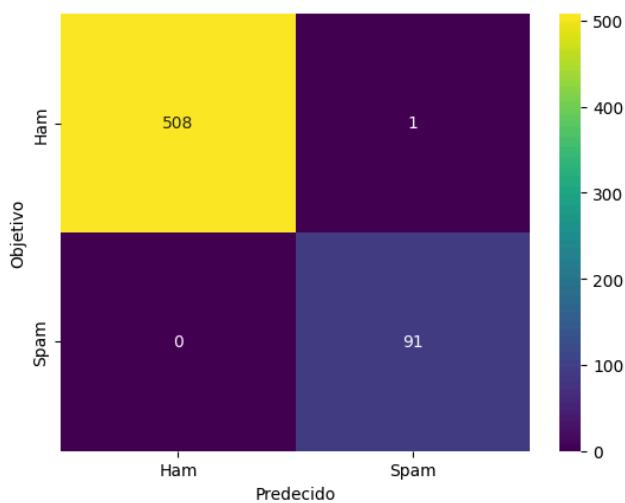
```
Out[... array([[508,    1],  
              [ 0,  91]], dtype=int64)
```

```
In [... # Gráfico de la matriz de confusión  
def plot_conf_matrix(c_m, color):  
    classes = ['Ham', 'Spam']  
    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 [... color = 'Blues' #coolwarm / viridis / Blues / Greens / Reds / magma / cividis  
plot_conf_matrix(conf_matrix_train, color)
```



```
In [...]: color = 'viridis'
plot_conf_matrix(conf_matrix_test, color)
```



La matriz de correlación, permite visualizar la cantidad de correos que se analizaron en el grupo de entrenamiento y de prueba, y se puede interpretar de la siguiente manera:

- La fila **0**, habla de los correos esperados que **spam**.

508	1
-----	---

- La fila **1**, habla de los correos esperados que son **spam**.

0	91
---	----

- La columna **0**, habla de los correos predichos que **no spam**.

508
0

- La columna **1**, habla de los correos predichos que son **spam**.

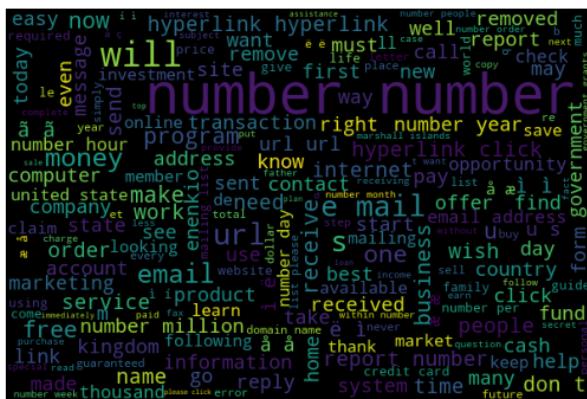
1

- La celda **(0, 0)**, muestra los correos que **no spam**, que se esperaban y que se predijeron correctamente.
 - La celda **(1, 1)**, muestra los correos que son **spam**, que se esperaban y que también se predijeron correctamente.
 - La celda **(0, 1)**, muestra los correos que se esperaban como **spam** y que la predicción los arrojó como **no spam**, es decir, los correos tipo **spam** que se lograron colar como correos buenos.
 - La celda **(1, 0)**, muestra los correos que se esperaban como **no spam** y que la predicción los arrojó como **spam**, es decir, los falsos **spam** positivos.

WordCloud

```
In [...] def visualize(label):
    words = ''
    for msg in df[df['spam'] == label]['contenido']:
        msg = msg.lower()
        words += msg + ' '
    wordcloud = WordCloud(width=600, height=400).generate(words)
    plt.imshow(wordcloud)
    plt.axis('off')
    plt.show()
```

In [...] visualize(1) #enviamos un 1, ya que la columna spam es 1 si el correo es spam



Explicación

Este generador de nube de palabras, se crea a partir de una cadena **words** vacía, la cual se va llenando cada vez que al recorrer el ciclo **for**, se notan con mayor frecuencia ciertas palabras en los mensajes clasificados como **spam**. Se convierten las palabras a minúsculas y se muestran utilizando el método **WordCloud** de la librería que lleva su mismo nombre.

Entre más se repite una palabra, más grande se ve en la nube de palabras.

Identificación de Falsos Spam

```
In [... # Vectorizamos la columna contenido
      X = vectores.transform(df['contenido'])
      # Creamos la columna de predicciones
      df['predicciones'] = model.predict(X)
      df
```

Out[...]

		contenido	spam	predicciones
0	date wed NUMBER aug NUMBER NUMBER NUMBER NUMB...	0	0	0
1	martin a posted tassos papadopoulos the greek ...	0	0	0
2	man threatens explosion in moscow thursday aug...	0	0	0
3	klez the virus that won t die already the most...	0	0	0
4	in adding cream to spaghetti carbonara which ...	0	0	0
...
2995	abc s good morning america ranks it the NUMBE...	1	1	1
2996	hyperlink hyperlink hyperlink let mortgage le...	1	1	1
2997	thank you for shopping with us gifts for all ...	1	1	1
2998	the famous ebay marketing e course learn to s...	1	1	1
2999	hello this is chinese traditional à¤à»í NUM...	1	1	1

3000 rows × 3 columns

In [...]

```
# identificación de los falsos positivos
falso_spam = df[(df['predicciones'] == 1) & (df['spam'] == 0)]['contenido']
falso_ham = df[(df['predicciones'] == 0) & (df['spam'] == 1)]['contenido']
if falso_spam.empty:
    print('No se encontraron falsos Spam')
else:
    print('**Falsos Spam**\n')
    for msg in falso_spam:
        print(msg[:300])

if falso_ham.empty:
    print('\n\nNo se encontraron falsos Ham')
else:
    print('\n\n**Falsos Ham**\n')
    for msg in falso_ham:
        print(msg[:10])
```

Falsos Spam

with our telecoms partner bumblebee don t get ripped off by expensive hotel payphone and mobile charges save save save on international calls with ryanair s phone partner you ll save up to NUMBER on international phone calls when you use our online phone card you can use the card from any phone in

url URL date not supplied detailed guidelines for vaccinating all NUMBER million citizens within five days of an outbreak are being dispatched to every state

Falsos Ham

NUMBER NU
x m a h c
this URL
r v r f i

r v r f i
r v r f i

this URL

Conclusiones

Se realizó un modelo basado en datos de correos electrónicos en español e inglés, obteniendo resultados diferentes pero con alta probabilidad de clasificación. Esto permite identificar que los modelos de Naive Bayes, en este caso el Multinomial, permiten realizar un óptimo proceso para separar correos basura.

Mg. Luis Felipe Bustamante Narváez

Anexo 12

Proyecto 12: Análisis de Sentimiento

Mg. Luis Felipe Bustamante Narváez

En este proyecto vamos a analizar unos dataset que contienen opiniones de usuarios y un análisis previo sobre dichas opiniones, si estas son positivas, negativas o neutras. El objetivo es predecir emociones o sentimientos de las personas, basado en los comentarios que dejan al respecto de un producto, situación o cualquier variable que apela a un calificativo.

Librerías

```
In [...]
import numpy as np
import pandas as pd
import seaborn as sn
import matplotlib.pyplot as plt
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score, f1_score, confusion_matrix
from sklearn.model_selection import train_test_split
import itertools
```

Cargamos los datos

```
In [...]
path = 'data/comentarios_peliculas.csv'
df = pd.read_csv(path, encoding='utf-8')
```

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

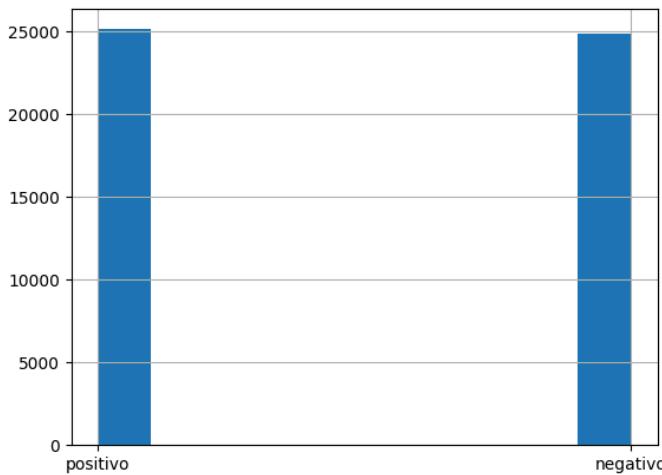
```
Out[...]
```

	comentario	sentimiento
0	Cada escena estaba llena de emoción y significó mucho para mí.	positivo
1	El guion es predecible y los actores no transmiten emociones.	negativo
2	Los efectos especiales eran ridículos y poco creativos.	negativo
3	Muy mala, esperaba mucho más y me decepcionó.	negativo
4	Una obra maestra, la mejor película que he visto en años.	positivo
...
49995	Cada escena estaba llena de emoción y significó mucho para mí.	positivo
49996	El guion es predecible y los actores no transmiten emociones.	negativo
49997	Una película increíble, me encantó cada escena.	positivo
49998	Terrible, no entiendo cómo alguien puede disfrutar de esta película.	negativo
49999	La historia es fascinante y los actores hicieron un gran trabajo.	positivo

50000 rows × 2 columns

```
In [...]
# histograma de sentimientos
df['sentimiento'].hist()
```

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



```
In [...] # Anexamos una columna binaria (objetivo)
target_map = {'positivo': 1, 'negativo': 0}
df['target'] = df['sentimiento'].map(target_map)
df
```

```
Out[...]      comentario  sentimiento  target
 0   Cada escena estaba llena de emoción y signific...  positivo    1
 1   El guion es predecible y los actores no transm...  negativo    0
 2   Los efectos especiales eran ridículos y poco c...  negativo    0
 3   Muy mala, esperaba mucho más y me decepcionó.  negativo    0
 4   Una obra maestra, la mejor película que he vis...  positivo    1
 ...
 49995  Cada escena estaba llena de emoción y signific...  positivo    1
 49996  El guion es predecible y los actores no transm...  negativo    0
 49997  Una película increíble, me encantó cada escena.  positivo    1
 49998  Terrible, no entiendo cómo alguien puede disfr...  negativo    0
 49999  La historia es fascinante y los actores hicier...  positivo    1
```

50000 rows × 3 columns

```
In [...] # Agrupamos para obtener el total de datos (0-> negativo, 1->positivo)
grouped = df.groupby('target').count()
grouped
```

```
Out[...]      comentario  sentimiento
  target
  0        24871       24871
  1        25129       25129
```

Procesamiento de los Datos

Entrenamiento

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

```
In [...] df_train
```

Out[...]

		comentario	sentimiento	target
34636	Me hizo reír, llorar y reflexionar, todo en un...	positivo	1	
34951	Los efectos especiales eran ridículos y poco c...	positivo	1	
27374	El desarrollo de los personajes es pobre y sin...	negativo	0	
32526	El final fue absurdo y dejó muchas preguntas s...	positivo	1	
45042	Una película increíble, me encantó cada escena.	positivo	1	
...	
10413	El final fue absurdo y dejó muchas preguntas s...	negativo	0	
9468	El guion es predecible y los actores no transm...	negativo	0	
11220	El desarrollo de los personajes es pobre y sin...	negativo	0	
9298	No tiene ni pies ni cabeza, simplemente mala.	negativo	0	
35665	El desarrollo de los personajes es pobre y sin...	negativo	0	

37500 rows × 3 columns

In [...]

df_test

Out[...]

		comentario	sentimiento	target
22896	El final fue absurdo y dejó muchas preguntas s...	positivo	1	
45709	Muy entretenida, sin duda la volvería a ver.	positivo	1	
12283	Una pérdida de tiempo total, me arrepiento de ...	negativo	0	
43490	Muy mala, esperaba mucho más y me decepcionó.	negativo	0	
4261	Me aburrí desde el primer minuto, no la recomi...	negativo	0	
...	
42442	Muy entretenida, sin duda la volvería a ver.	positivo	1	
27913	El desarrollo de los personajes es pobre y sin...	negativo	0	
38611	El guion es predecible y los actores no transm...	negativo	0	
7900	Una película increíble, me encantó cada escena.	positivo	1	
14261	El guion es predecible y los actores no transm...	negativo	0	

12500 rows × 3 columns

Vectorización

In [...]

```
# usamos un máximo de 2000 dimensiones
vectorizer = TfidfVectorizer(max_features=2000)
```

In [...]

```
# vectorizamos el entrenamiento
X_train = vectorizer.fit_transform(df_train['comentario'])
X_train
```

Out[...]

```
<37500x109 sparse matrix of type '<class 'numpy.float64'>' 
 with 305339 stored elements in Compressed Sparse Row format>
```

In [...]

```
X_test = vectorizer.transform(df_test['comentario'])
X_test
```

Out[...]

```
<12500x109 sparse matrix of type '<class 'numpy.float64'>' 
 with 101755 stored elements in Compressed Sparse Row format>
```

In [...]

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

In [...]

```
len(Y_train)
```

Out[...]

```
37500
```

In [...]

```
len(Y_test)
```

```
Out[... 12500
```

Modelo

```
In [... model = LogisticRegression(max_iter=500) #Genera 500 iteraciones cambiando los pesos/sesgos  
model.fit(X_train, Y_train)
```

```
Out[... ▾ LogisticRegression ⓘ ?  
LogisticRegression(max_iter=500)
```

```
In [... # Probamos el modelo con los datos originales  
train_accuracy = model.score(X_train, Y_train)  
test_accuracy = model.score(X_test, Y_test)
```

```
In [... # Mostramos la puntuación  
print(f'El accuracy de entrenamiento es de {train_accuracy}')  
print(f'El accuracy de prueba es de {test_accuracy}')
```

```
El accuracy de entrenamiento es de 0.95088  
El accuracy de prueba es de 0.94736
```

Predicciones

```
In [... P_train = model.predict(X_train)  
P_test = model.predict(X_test)
```

```
In [... # Matriz de confusión  
conf_matrix_train = confusion_matrix(Y_train, P_train, normalize='true')  
conf_matrix_train
```

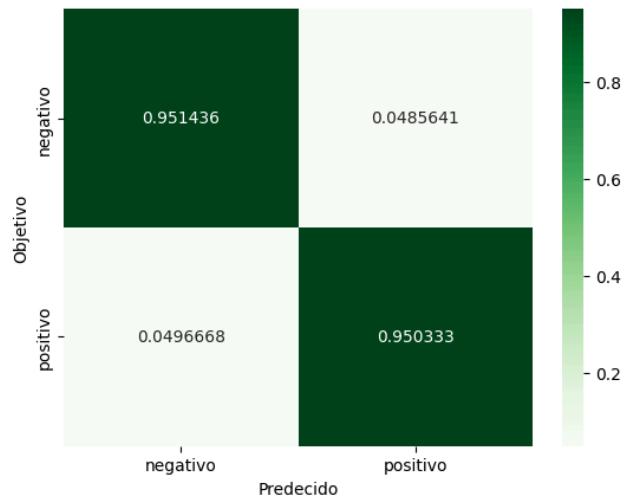
```
Out[... array([[0.95143595, 0.04856405],  
              [0.04966677, 0.95033323]])
```

```
In [... conf_matrix_test = confusion_matrix(Y_test, P_test, normalize='true')  
conf_matrix_test
```

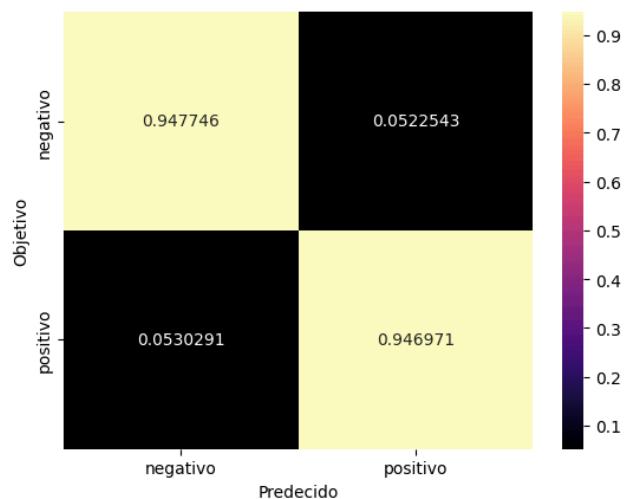
```
Out[... array([[0.94774574, 0.05225426],  
              [0.05302909, 0.94697091]])
```

```
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 [... color = 'Greens' #coolwarm / viridis / Blues / Greens / Reds / magma / cividis  
plot_conf_matrix(conf_matrix_train, color)
```



```
In [...] color = 'magma'
plot_conf_matrix(conf_matrix_test, color)
```



Análisis de las palabras

```
In [...] # vectorización de palabras
word_index_map = vectorizer.vocabulary_
dict(itertools.islice(word_index_map.items(), 5))
```

```
Out[...]: {'me': 56, 'hizo': 45, 'reír': 87, 'llorar': 51, 'reflexionar': 85}
```

```
In [...] # Coeficiente de las palabras
model.coef_[0][:10]
```

```
Out[...]: array([-1.44036264e+00, -1.15267244e+00, 4.48615605e-04, 6.22903789e-01,
-1.09387316e+00, -1.21140622e+00, -8.04303645e-01, 5.58432574e-01,
-7.37314123e-01, 2.10564178e+00])
```

```
In [...] limit = 1.5
print('**Palabras más positivas**\n')
for word, index in word_index_map.items():
    weight = model.coef_[0][index]
    if weight > limit:
        print(word, weight)
```

```
**Palabras más positivas**
```

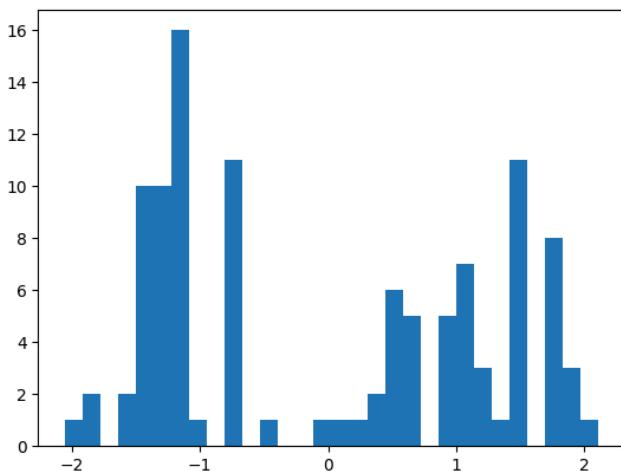
```
cada 2.1056417823800224
escena 1.8834508871239404
perfecto 1.7561162834981257
podría 1.7561162834981257
haber 1.7561162834981257
sido 1.7561162834981257
mejor 1.9163405384954952
la 1.7578370103429033
emoción 1.5203371110754182
excelente 1.7906494248075528
visuales 1.7906494248075528
impresionantes 1.7906494248075528
un 1.939566627288709
```

```
In [... limit = 1.5
print('**Palabras más negativas**\n')
for word, index in word_index_map.items():
    weight = model.coef_[0][index]
    if weight < -limit:
        print(word, weight)

**Palabras más negativas**
```

```
el -1.7793619395843912
de -1.8047724408415342
no -1.571073260466105
mala -2.051899328241801
esta -1.5697751259113608
```

```
In [... # Gráfico de pesos
plt.hist(model.coef_[0], bins=30)
plt.show()
```



Probamos el modelo

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

```
In [... # Vectorizamos la prueba
x = vectorizer.transform(prueba)
```

```
x  
Out[... <3x109 sparse matrix of type '<class 'numpy.float64'>'  
       with 12 stored elements in Compressed Sparse Row format>  
In [... # Predecimos con el modelo  
P = model.predict(x)  
In [... # Obtenemos las clases del modelo  
clases = model.classes_  
In [... # Mostramos la clase de la prueba  
for i in range(len(prueba)):  
    if clases[P[i]] == 0:  
        print(f'El comentario "{prueba[i]}" es Negativo')  
    else:  
        print(f'El comentario "{prueba[i]}" es Positivo')  
El comentario "estuvo muy entretenida la película" es Positivo  
El comentario "estuvo horrible la película, me aburrió mucho" es Negativo  
El comentario "no la recomiendo" es Negativo
```

Conclusiones

Se realizó un modelo basado en datos de comentarios sobre películas, obteniendo resultados diferentes pero con alta probabilidad de clasificación. Esto permite identificar que la función Sigmoid permiten realizar un óptimo proceso para separar reconocer sentimientos positivos y negativos.

Mg. Luis Felipe Bustamante Narváez

Anexo 13

Proyecto 13: Análisis de Sentimiento Multiclasificación

Mg. Luis Felipe Bustamante Narváez

En este proyecto vamos a analizar unos dataset que contienen opiniones de usuarios y un análisis previo sobre dichas opiniones, si estas son positivas, negativas o neutras. El objetivo es predecir emociones o sentimientos de las personas, basado en los comentarios que dejan en redes sociales, para el ejemplo, facebook.

A diferencia del Proyecto 12, en este caso, usaremos multiclasificación, de tal manera que se puedan analizar más de dos sentimientos, y el proceso deje de ser binario.

Librerías

```
In [...]
import numpy as np
import pandas as pd
import seaborn as sn
import matplotlib.pyplot as plt
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score, f1_score, confusion_matrix
from sklearn.model_selection import train_test_split
import itertools
```

Cargamos los datos

```
In [...]
path = 'data/comentarios_facebook.csv'
df = pd.read_csv(path, encoding='utf-8')
```

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

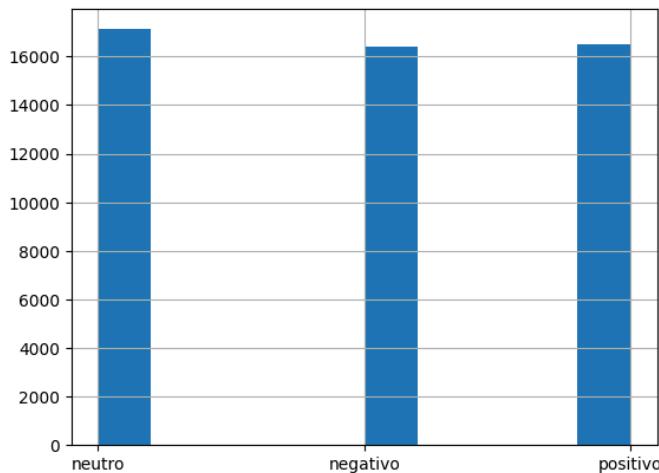
```
Out[...]
```

	comentario	sentimiento
0	No afecta mi vida, pero es interesante saberlo.	neutro
1	Esto es una completa pérdida de tiempo.	negativo
2	Buenísima recomendación, la probaré.	positivo
3	Es un buen punto, aunque depende de la perspec...	neutro
4	Qué publicación tan aburrida y sin sentido.	negativo
...
49995	Me encanta esta publicación, muy inspiradora!	neutro
49996	Me decepcionó mucho este contenido.	negativo
49997	Ni bueno ni malo, simplemente un comentario más.	neutro
49998	No entiendo por qué esto está en mi feed.	negativo
49999	No estoy seguro de qué pensar al respecto.	neutro

50000 rows × 2 columns

```
In [...]
# histograma de sentimientos
df['sentimiento'].hist()
```

```
Out[...]
<Axes: >
```



```
In [...] # Anexamos una columna binaria (objetivo)
target_map = {'positivo': 1, 'negativo': 0, 'neutro': 2}
df['target'] = df['sentimiento'].map(target_map)
df
```

```
Out[...]      comentario  sentimiento  target
0           No afecta mi vida, pero es interesante saberlo.    neutro     2
1           Esto es una completa pérdida de tiempo.    negativo     0
2           Buenísima recomendación, la probaré!    positivo     1
3           Es un buen punto, aunque depende de la perspec...    neutro     2
4           Qué publicación tan aburrida y sin sentido.    negativo     0
...
49995       Me encanta esta publicación, muy inspiradora!    neutro     2
49996       Me decepcionó mucho este contenido.    negativo     0
49997       Ni bueno ni malo, simplemente un comentario más.    neutro     2
49998       No entiendo por qué esto está en mi feed.    negativo     0
49999       No estoy seguro de qué pensar al respecto.    neutro     2
```

50000 rows × 3 columns

```
In [...] # Agrupamos para obtener el total de datos (0-> negativo, 1->positivo)
grouped = df.groupby('target').count()
grouped
```

```
Out[...]      comentario  sentimiento
target
0           16384        16384
1           16495        16495
2           17121        17121
```

Procesamiento de los Datos

Entrenamiento

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

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

		comentario	sentimiento	target
46633	Interesante, pero me gustaría más información.	neutro	2	
14472	Gran trabajo, sigue así!	positivo	1	
19520	Ni bueno ni malo, simplemente un comentario más.	neutro	2	
1674	Podría ser cierto, aunque hay otras opiniones.	neutro	2	
27658	Es un buen punto, aunque depende de la perspec...	positivo	1	
...	
6088	Algunas partes son ciertas, otras no tanto.	neutro	2	
24509	Me hiciste reír, gracias por compartir!	positivo	1	
32483	Me hiciste reír, gracias por compartir!	positivo	1	
42789	Me decepcionó mucho este contenido.	negativo	0	
43208	Es un buen punto, aunque depende de la perspec...	neutro	2	

37500 rows × 3 columns

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

```
df_test
```

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

		comentario	sentimiento	target
1745	Tu contenido ha ido empeorando cada vez más.	negativo	0	
5583	Demasiado exagerado, no vale la pena leerlo.	negativo	0	
47317	Tu contenido ha ido empeorando cada vez más.	negativo	0	
457	Malísima recomendación, no la sigan.	negativo	0	
31703	Me decepcionó mucho este contenido.	negativo	0	
...	
8044	No estoy de acuerdo para nada, pésimo argumento.	negativo	0	
21421	No entiendo por qué esto está en mi feed.	negativo	0	
38151	Ni bueno ni malo, simplemente un comentario más.	neutro	2	
4594	No tiene sentido lo que dices, muy decepcionante.	negativo	0	
4596	Me decepcionó mucho este contenido.	negativo	0	

12500 rows × 3 columns

Vectorización

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

```
# usamos un máximo de 2000 dimensiones
vectorizer = TfidfVectorizer(max_features=2000)
```

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

```
# vectorizamos el entrenamiento
X_train = vectorizer.fit_transform(df_train['comentario'])
X_train
```

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

```
<37500x132 sparse matrix of type '<class 'numpy.float64'>'  
with 256951 stored elements in Compressed Sparse Row format>
```

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

```
X_test = vectorizer.transform(df_test['comentario'])
X_test
```

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

```
<12500x132 sparse matrix of type '<class 'numpy.float64'>'  
with 85272 stored elements in Compressed Sparse Row format>
```

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

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

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

```
len(Y_train)
```

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

```
37500
```

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

```
len(Y_test)
```

```
Out[...]: 12500
```

Modelo

```
In [...]: model = LogisticRegression(max_iter=1000) #Genera 1000 iteraciones cambiando los pesos/sesgos  
model.fit(X_train, Y_train)
```

```
Out[...]: LogisticRegression(max_iter=1000)
```

```
In [...]: # Probamos el modelo con los datos originales  
train_accuracy = model.score(X_train, Y_train)  
test_accuracy = model.score(X_test, Y_test)
```

```
In [...]: # Mostramos la puntuación  
print(f'El accuracy de entrenamiento es de {train_accuracy}')  
print(f'El accuracy de prueba es de {test_accuracy}')
```

```
El accuracy de entrenamiento es de 0.9496533333333333  
El accuracy de prueba es de 0.95104
```

Predicciones

```
In [...]: P_train = model.predict(X_train)  
P_test = model.predict(X_test)
```

```
In [...]: # Matriz de confusión  
conf_matrix_train = confusion_matrix(Y_train, P_train, normalize='true')  
conf_matrix_train
```

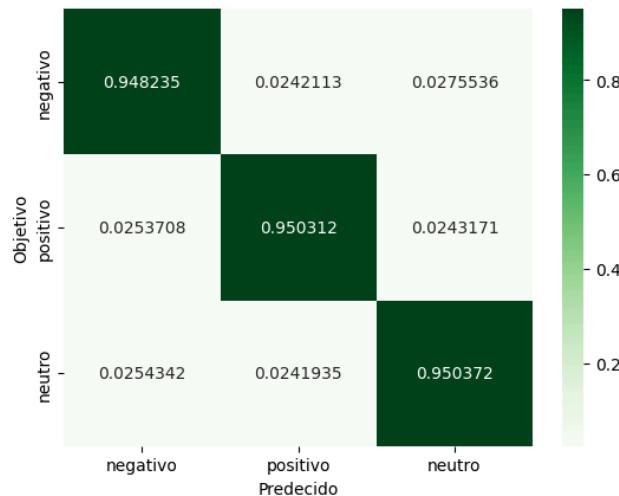
```
Out[...]: array([[0.9482351 , 0.0242113 , 0.0275536 ],  
                 [0.02537084, 0.95031207, 0.02431709],  
                 [0.02543424, 0.02419355, 0.95037221]])
```

```
In [...]: conf_matrix_test = confusion_matrix(Y_test, P_test, normalize='true')  
conf_matrix_test
```

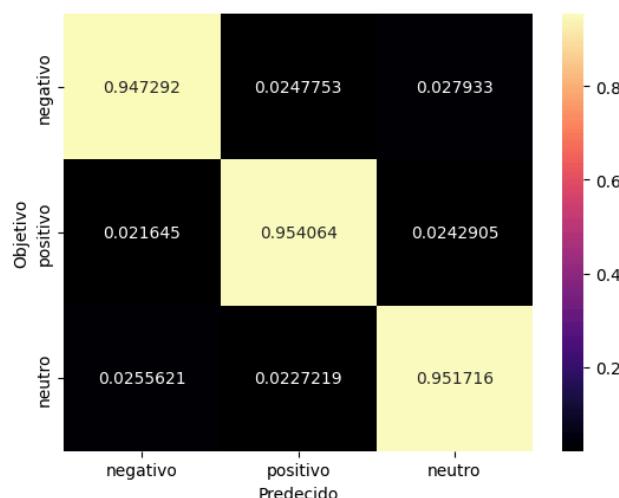
```
Out[...]: array([[0.94729172, 0.02477532, 0.02793296],  
                 [0.02164502, 0.95406445, 0.02429052],  
                 [0.02556213, 0.02272189, 0.95171598]])
```

```
In [...]: # Gráfico de la matriz de confusión  
def plot_conf_matrix(c_m, color):  
    classes = ['negativo', 'positivo', 'neutro']  
    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 [...]: color = 'Greens' #coolwarm / viridis / Blues / Greens / Reds / magma / cividis  
plot_conf_matrix(conf_matrix_train, color)
```



```
In [...] color = 'magma'
plot_conf_matrix(conf_matrix_test, color)
```



Análisis de las palabras

```
In [...] # vectorización de palabras
word_index_map = vectorizer.vocabulary_
dict(itertools.islice(word_index_map.items(), 5))
```

```
Out[... {'interesante': 61, 'pero': 87, 'me': 69, 'gustaría': 51, 'más': 76}
```

```
In [...] # Coeficiente de las palabras
model.coef_[0].max()
```

```
Out[... 2.1007847353587525
```

```
In [...] limit = 1.5
print('**Palabras más negativas**\n')
for word, index in word_index_map.items():
    weight = model.coef_[0][index]
    if weight > limit:
        print(word, weight)
```

```
**Palabras más negativas**
```

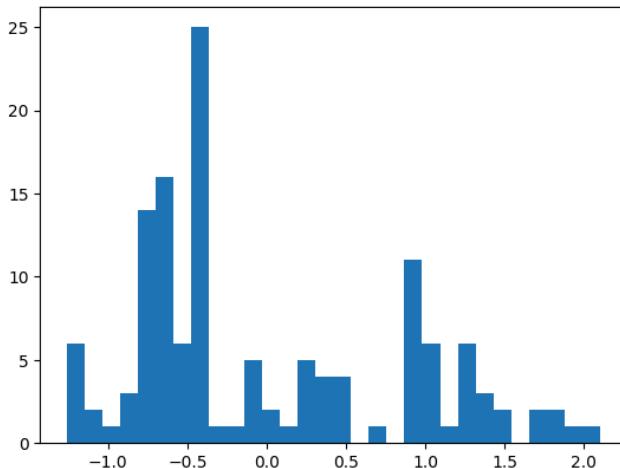
```
no 1.5109386310454178
esto 2.1007847353587525
deception 1.8484200261758081
este 1.8484200261758081
sentido 1.9737735487852108
malísima 1.6576414288486296
sigan 1.6576414288486296
```

```
In [... limit = 1.1
print('**Palabras más positivas**\n')
for word, index in word_index_map.items():
    weight = model.coef_[0][index]
    if weight < -limit:
        print(word, weight)

**Palabras más positivas**
```

```
pero -1.1952465233388108
buenísima -1.2596535749701068
probaré -1.2596535749701068
excelente -1.1692834161609396
al -1.1574146905115645
respecto -1.1574146905115645
```

```
In [... # Gráfico de pesos
plt.hist(model.coef_[0], bins=30)
plt.show()
```



Probamos el modelo

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

```
In [... # Vectorizamos la prueba
x = vectorizer.transform(prueba)
x
```

```
Out[... <3x132 sparse matrix of type '<class 'numpy.float64'>'
      with 7 stored elements in Compressed Sparse Row format>
```

```
In [...] # Predecimos con el modelo  
P = model.predict(x)  
  
In [...] # Obtenemos las clases del modelo  
clases = model.classes_  
  
In [...] # Mostramos la clase de la prueba  
for i in range(len(prueba)):  
    if clases[P[i]] == 0:  
        print(f'El comentario "{prueba[i]}" es Negativo')  
    elif clases[P[i]] == 2:  
        print(f'El comentario "{prueba[i]}" es Neutro')  
    else:  
        print(f'El comentario "{prueba[i]}" es Positivo')  
  
El comentario "estuve muy entretenida la película" es Positivo  
El comentario "estuve horrible la película, me aburrió mucho" es Positivo  
El comentario "no la recomiendo" es Negativo
```

Conclusiones

Se realizó un modelo basado en datos de comentarios en redes sociales, obteniendo resultados diferentes pero con alta probabilidad de clasificación. Esto permite identificar que la función Softmax permiten realizar un óptimo proceso para separar y reconocer sentimientos positivos y negativos y de otras características..

Mg. Luis Felipe Bustamante Narváez

Anexo 14

Proyecto 14: Resumen de textos con Vectorización

Mg. Luis Felipe Bustamante Narváez

En este proyecto, desarrollaremos un generador de resúmenes a través de vectorización, tomando una base de datos importante, de diferentes artículos de prensa para analizar cómo pueden sumarizarse y permitir los beneficios de la inteligencia artificial.

Librerías

```
In [...] import pandas as pd  
import numpy as np  
import textwrap  
import nltk  
from nltk.corpus import stopwords  
from nltk import word_tokenize  
from sklearn.feature_extraction.text import TfidfVectorizer
```

```
In [...] nltk.download('punkt')  
nltk.download('stopwords')  
  
[nltk_data] Downloading package punkt to  
[nltk_data]     C:\Users\luis_\AppData\Roaming\nltk_data...  
[nltk_data]   Package punkt is already up-to-date!  
[nltk_data] Downloading package stopwords to  
[nltk_data]     C:\Users\luis_\AppData\Roaming\nltk_data...  
[nltk_data]   Package stopwords is already up-to-date!
```

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

```
In [...] path = 'data/df_total.csv'  
df = pd.read_csv(path, encoding='utf-8')
```

```
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

```
In [...] print(df['news'][2][:500])
```

En una industria históricamente masculina como lo es la aviación Viva presentó su avión rosado A320NEO que apuesta por la equidad de género la lucha contra el cáncer de mama la inclusión y la diversidad. Desde Francia llegó Go Pink que tuvo un precio promedio de US\$50 millones convirtiéndose en la aeronave número 20 de las 21 con las que finalizará el año esta aerolínea. En Viva estamos trabajando muy fuerte para que haya más mujeres. Actualmente el grupo ejecutivo está compuesto por 42 mujeres.

```
In [...] # Buscamos una noticia larga para tomar como ejemplo a La hora de hacer el resumen  
doc = df['news'].sample()
```

```
In [...] print(doc.iloc[0][:500])
```

El actual brote de inflación es un momento de déjà vu para las personas que vivieron las subidas de precios de principios de la década de 1980. La inflación de EE.UU. se aceleró a una tasa anual de 7,5% en enero alcanzando un máximo de cuatro décadas. El índice de precios al consumidor que mide lo que la gente paga por bienes y servicios estuvo el mes pasado en su nivel más alto desde febrero de 1982 en comparación con enero de hace un año según el Departamento de Trabajo. Blaise Jones recuerda a su

```
In [...] # Obtener el índice de la noticia para manipulación de datos  
indice = df.index[df['news'] == doc.iloc[0]].tolist()  
print(indice)
```

```
[297]
```

```
In [...] # hacemos la prueba  
print(df['news'][297][:100])
```

El actual brote de inflación es un momento de déjà vu para las personas que vivieron las subidas de

Procesamiento de Datos

TextWrap

```
In [...] # Eliminamos las palabras cortadas de las líneas  
doc2 = textwrap.fill(doc.iloc[0], replace_whitespace=False, fix_sentence_endings=True)
```

```
In [...] print(doc2[:500])
```

El actual brote de inflación es un momento de déjà vu para las personas que vivieron las subidas de precios de principios de la década de 1980. La inflación de EE.UU. se aceleró a una tasa anual de 7,5% en enero alcanzando un máximo de cuatro décadas. El índice de precios al consumidor que mide lo que la gente paga por bienes y servicios estuvo el mes pasado en su nivel más alto desde febrero de 1982 en comparación con enero de hace un año según el Departamento de Trabajo. Blaise Jones recuerda a su

Separación de líneas

```
In [...] # Podemos separar por líneas, por puntos o comas, la idea es conservar oraciones  
# con ideas claras.  
lineas = doc2.split('. ') # Usamos punto espacio, por las siglas o números que pueden haber  
lineas[:3]
```

```
Out[...]: ['El actual brote de inflación es un momento de déjà vu para las\npersonas que vivieron las subidas de precios de principios de la\ndécada de 1980. La inflación de EE.UU', 'se aceleró a una tasa anual de\n75 en enero alcanzando un máximo de cuatro décadas', 'El índice de\nprecios al consumidor que mide lo que la gente paga por bienes y\nservicios estuvo el mes pasado en su nivel más alto desde febrero de\n1982 en comparación con enero de hace un año según el Departamento de\nTrabajo. Blaise Jones recuerda a su madre hablando sobre el aumento del\nprecio de la leche y la determinación de su padre de mantener baja la\nfacción de calefacción de su hogar tácticas que incluían bajar el\ntermostato a 62 grados a la hora de acostarse. Juro que podía ver mi\nrespiración cuando me levantaba dijo el doctor Jones ahora de 59 años\ny neurorradiólogo pediatrónico en Cincinnati']
```

Tokenización

```
In [...]: # Creamos la tokenización usando las stopwords descargadas
tokenizar = TfidfVectorizer(stop_words=stopwords.words('spanish'), norm='l1')
```

```
In [...]: # Creamos la matriz
X = tokenizar.fit_transform(lineas)
X
```

```
Out[...]: <26x433 sparse matrix of type '<class 'numpy.float64'>' with 560 stored elements in Compressed Sparse Row format>
```

```
In [...]: # Mostramos la matriz
filas, columnas = X.shape

for i in range(10): # aquí ponemos las filas, pero al ser muchas el resultado es extenso.
    for j in range(10):
        print(X[i, j], end=' ') # imprime el elemento y un espacio en blanco
    print() #deja el renglón

# Cada fila va a representar una palabra
# Cada columna va a representar cada una de las oraciones
```

```
0.0 0.0 0.0 0.0 0.0 0.04756870737569473 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.018470715419332654 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.08059297416517343 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
```

```
In [...]: # Promediamos los puntajes de cada una de las oraciones
def obtener_score(tfidf_row):
    x = tfidf_row[tfidf_row != 0] # Elimina las oraciones que no tienen puntuación
    return x.mean()
```

```
In [...]: # Creamos el vector de puntuación y lo llenamos
scores = np.zeros(len(lineas))
for i in range(len(lineas)):
    score = obtener_score(X[i,:])
    scores[i] = score
```

```
In [...]: scores
```

```
Out[...]: array([0.06666667, 0.11111111, 0.01754386, 0.05      , 0.07692308,
   0.0625    , 1.        , 0.25      , 0.04761905, 0.03703704,
   0.04545455, 0.02439024, 0.03846154, 0.02083333, 0.06666667,
   0.03225806, 0.04545455, 0.0625    , 0.05263158, 0.03225806,
   0.125     , 0.09090909, 0.03703704, 0.07692308, 0.03030303,
   0.07142857])
```

Resumen

```
In [...]: # Ordenamos los scores y mostramos las posiciones de mayor a menor
sort_index = np.argsort(-scores)
```

```
In [...]: sort_index
```

```
Out[...]: array([ 6,  7, 20,  1, 21, 23,  4, 25, 14,  0, 17,  5, 18,  3,  8, 10, 16,
   12,  9, 22, 15, 19, 24, 11, 13,  2], dtype=int64)
```

```
In [...]: # Resumen desordenado
oraciones = []
cantidad_oraciones = 10
for i in range(cantidad_oraciones):
    oraciones.append([sort_index[i], scores[sort_index[i]], lineas[sort_index[i]]])
print(f'{scores[sort_index[i]]}:\n{lineas[sort_index[i]]}\n')
```

1.0:
Jones

0.25:
Él y su esposa han sido frugales durante mucho tiempo

0.125:
Está posponiendo la instalación de un nuevo revestimiento de aluminio en su casa porque los precios han subido

0.1111111111111111:
se aceleró a una tasa anual de 75 en enero alcanzando un máximo de cuatro décadas

0.09090909090909093:
Pagó el préstamo de su automóvil hace unos dos años pero continúa conduciendo un BMW 2014 golpeado

0.07692307692307694:
El fundador de la empresa de ferias comerciales Shamrock Productions condujo su Oldsmobile durante 450.000 millas hasta que el motor explotó

0.07692307692307694:
dos veces por semana para llenar la camioneta de la familia y tratar de evitar las colas en la bomba durante la crisis energética de 1979

0.07142857142857144:
Ella también se está saltando vacaciones costosas por ahora.A cada paso la gente se ve afectada por el aumento de los precios dijo la señora Navratil.

0.06666666666666667:
Cuando cerró la compra de la casa la tasa hipotecaria que el corredor había ofrecido saltó a cerca de 135 desde alrededor de 1275 que tenía cuando había iniciado el proceso dijo

0.06666666666666667:
El actual brote de inflación es un momento de déjà vu para las personas que vivieron las subidas de precios de principios de la década de 1980.La inflación de EE.UU

```
In [...] # Ordenamiento de la lista por el primer elemento de cada sublista
oraciones_sort = sorted(oraciones, key=lambda x:x[0])

#Imprimimos la lista ordenada
for item in oraciones_sort:
    print(item[2]) # el 2 es la columna de las líneas
```

El actual brote de inflación es un momento de déjà vu para las personas que vivieron las subidas de precios de principios de la década de 1980. La inflación de EE.UU se aceleró a una tasa anual de 7,5% en enero alcanzando un máximo de cuatro décadas dos veces por semana para llenar la camioneta de la familia y tratar de evitar las colas en la bomba durante la crisis energética de 1979.

Jones

Él y su esposa han sido frugales durante mucho tiempo.

Cuando cerró la compra de la casa la tasa hipotecaria que el corredor había ofrecido saltó a cerca de 13,5% desde alrededor de 12,75% que tenía cuando había iniciado el proceso.

dijo

Está posponiendo la instalación de un nuevo revestimiento de aluminio en su casa porque los precios han subido.

Pagó el préstamo de su automóvil hace unos dos años pero continúa conduciendo un BMW 2014 golpeado.

El fundador de la empresa de ferias comerciales Shamrock Productions condujo su Oldsmobile durante 450.000 millas hasta que el motor explotó.

Ella también se está saltando vacaciones costosas por ahora. A cada paso la gente se ve afectada por el aumento de los precios dijo la señora Navratil.

Conclusiones

Hace algunos años, la vectorización era la manera más adecuada para generar resúmenes, como podemos notar en los resultados, hace falta un poco de coherencia, pero se puede entender la idea del texto que se pretende resumir.

Mg. Luis Felipe Bustamante Narváez

Anexo 15

Proyecto 15: Resumen de textos con Text Rank

Mg. Luis Felipe Bustamante Narváez

En este ejercicio, desarrollaremos un generador de resumen de textos avanzado, utilizando el método TextRank, el cual permite analizar la similitud de las oraciones y palabras para contextualizar las ideas principales del texto original.

Recordemos que el TextRank está basado en el PageRank de Google, donde las oraciones o palabras equivalen a las páginas web y la similitud entre las oraciones o palabras, representan los enlaces que estas páginas tienen con otras páginas importantes.

Librerías

```
In [...] pip install networkx
In [...] import pandas as pd
import numpy as np
import textwrap
import nltk
from nltk.corpus import stopwords
from nltk import word_tokenize
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
import textwrap
import networkx as nx
import matplotlib.pyplot as plt
In [...] nltk.download('punkt')
nltk.download('stopwords')

[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\luis_\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\luis_\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
Out[...]
```

Cargamos los Datos

```
In [...] path = 'data/df_total.csv'
df = pd.read_csv(path, encoding='utf-8')
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

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

```
print(df['news'][2][:500])
```

En una industria históricamente masculina como lo es la aviación Viva presentó su avión rosado A320NEO que apuesta por la equidad de género la lucha contra el cáncer de mama la inclusión y la diversidad. Desde Francia llegó Go Pink que tuvo un precio promedio de US\$50 millon es convirtiéndose en la aeronave número 20 de las 21 con las que finalizará el año esta aerolínea. En Viva estamos trabajando muy fuerte para que haya más mujeres. Actualmente el grupo ejecutivo está compuesto por 42 mujeres per

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

```
# Buscamos una noticia Larga para tomar como ejemplo a La hora de hacer el resumen  
doc = df['news'].sample()
```

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

```
print(doc.iloc[0][:500])
```

El crecimiento global será en los próximos dos años levemente más bajo a lo esperado pero se mantendrá robusto. Tras una caída del 3% en 2020 se estima que el PIB mundial creció alrededor de 6% en 2021 y que se expandirá 4% en 2022 y 3% en 2023. El informe de BBVA Research Situación Argentina 1 trimestre 2022 sostiene que la persistencia de la pandemia y de los problemas en las cadenas de suministro globales además de frenar el crecimiento económico mantendrán la inflación global elevada principa

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

```
# Obtener el índice de La noticia para manipulación de datos  
indice = df.index[df['news'] == doc.iloc[0]].tolist()  
print(indice)
```

```
[804]
```

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

```
# hacemos La prueba con un texto que tiene bastante texto  
print(df['news'][257][:100])
```

Como cambiarle a un coche su motor diésel por uno eléctrico mientras está en marcha. Esta analogía d

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

```
#Buscamos el texto Largo que encontramos en una fase de pruebas  
doc = df.loc[257, 'news']
```

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

```
doc[:500]
```

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

'Como cambiarle a un coche su motor diésel por uno eléctrico mientras está en marcha. Esta analogía define la tarea titánica que ha supuesto para Ethereum su última actualización. Frustrado de un trabajo de varios años, 'The Merge' constituye un cambio de paradigma sobre cómo se generan los criptoactivos de forma más segura y sostenible (elimina hasta un 99,95% de la energía necesaria hasta ahora), y permitirá que las finanzas descentralizadas (DeFi) afronten la implantación a gran escala. Pero tambi'

Procesamiento de Datos

TextWrap

```
In [... # Eliminamos Las palabras cortadas de Las Líneas
doc2 = textwrap.fill(doc, replace_whitespace=False, fix_sentence_endings=True)
```

```
In [... print(doc2[:500])
```

Como cambiarle a un coche su motor diésel por uno eléctrico mientras está en marcha. Esta analogía define la tarea titánica que ha supuesto para Ethereum su última actualización. Fruto de un trabajo de varios años, ‘The Merge’ constituye un cambio de paradigma sobre cómo se generan los criptoactivos de forma más segura y sostenible (elimina hasta un 99,95% de la energía necesaria hasta ahora), y permitirá que las finanzas descentralizadas (DeFi) afronten la implantación a gran escala. Pero ta

Separación en líneas

```
In [... # Podemos separar por Líneas, por puntos o comas, La idea es conservar oraciones
# con ideas claras.
lineas = doc2.split('. ') #Usamos punto espacio, por Las siglas o números que pueden haber
```

```
In [... len(lineas)
```

```
Out[... 24
```

```
In [... lineas[22:]
```

```
Out[... [' Pero ahora\nGary Gensler, presidente de la SEC (la Comisión de Bolsa y Valores de\nEstados Unidos), ha apuntado a la posibilidad de categorizarlo como un\nvalor, ya que con la fórmula del PoS se comporta como tal (genera\n dividendos a través del depósito), lo que lo sometería a las\nnormativas que regulan los valores.\r\nBrett Harrison, presidente de la\nplataforma de criptoactivos FTX.US, ha recordado que esta complejidad\naumenta al existir diferentes tipos de participación',
' Un usuario\npuede depositar sus ‘tokens’ para convertirse en validador, participar\nde manera indirecta a través de un ‘exchange’, como se ha mencionado\nanteriormente, o prestarlos en protocolos financieros descentralizados\nen los que obtener también rendimiento s.\r\nEl debate está sobre la\nmesa.]
```

Eliminación de oraciones vacías

```
In [... lineas = [item for item in lineas if item.strip()]
```

```
In [... len(lineas)
```

```
Out[... 24
```

```
In [... lineas[22:]
```

```
Out[...]: ['Pero ahora\nGary Gensler, presidente de la SEC (la Comisión de Bolsa y Valores de\nEstados Unidos), ha apuntado a la posibilidad de categorizarlo como un\nvalor, ya que con la fórmula del PoS se comporta como tal (genera\n dividendos a través del depósito), lo que lo sometería a las\nnormativas que regulan los valores.\r\nBrett Harrison, presidente de la\nplataforma de criptoactivos FTX.US, ha recordado que esta complejidad\naumenta al existir diferentes tipos de participación',\n 'Un usuario\npuede depositar sus 'tokens' para convertirse en validador, participar\nde manera indirecta a través de un 'exchange', como se ha mencionado\nanteriormente, o prestarlos en protocolos financieros descentralizados\nen los que obtener también rendimiento s.\r\nEl debate está sobre la\nmesa.']
```

Vectorización

```
In [...] # Creamos la tokenización usando las stopwords descargadas  
tokenizar = TfidfVectorizer(stop_words=stopwords.words('spanish'), norm='l1')
```

```
In [...]: # Creamos la matriz  
X = tokenizar.fit_transform(lineas)  
X
```

```
Out[...]: <24x402 sparse matrix of type '<class 'numpy.float64'>'  
        with 539 stored elements in Compressed Sparse Row format>
```

```
In [...] # Mostramos la matriz
filas, columnas = X.shape

for i in range(10): #aquí ponemos las filas, pero al ser muchas el resultado es extenso.
    for j in range(10):
        print(X[i, j], end=' ') # imprime el elemento y un espacio en blanco
    print() #deja el renglón

# Cada fila va a representar una palabra
# Cada columna va a representar cada una de las oraciones
```

Calculamos la similitud

```
In [...]: # Matriz de similitud  
S = cosine_similarity(X)
```

```
In [...]: s.shape
```

Out[...](24, 24)

```
In [...] # Muestra de similitudes, por ejemplo la oración 0, tiene similitud de 0.1248 con La  
# oración 6.  
S[:1]
```

```
Out[... array([[1.          , 0.          , 0.          , 0.          , 0.          ,
   0.          , 0.12486586, 0.          , 0.          , 0.12170713,
   0.07641065, 0.          , 0.          , 0.          , 0.          ,
   0.          , 0.          , 0.          , 0.          , 0.          ,
   0.          , 0.          , 0.          , 0.          ]])
```

Normalización

```
In [... S = S / S.sum(axis=1, keepdims=True)
```

```
In [... S[:1]
```

```
Out[... array([[0.75586724, 0.          , 0.          , 0.          , 0.          ,
   0.          , 0.09438202, 0.          , 0.          , 0.09199444,
   0.05775631, 0.          , 0.          , 0.          , 0.          ,
   0.          , 0.          , 0.          , 0.          , 0.          ,
   0.          , 0.          , 0.          , 0.          ]])
```

```
In [... # Probamos
S[0].sum()
```

```
Out[... 1.0
```

Suavizado (Markov)

```
In [... # Matriz de transición uniforme
U = np.ones_like(S)
```

```
In [... U[:1]
```

```
Out[... array([[1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
   1., 1., 1., 1., 1., 1., 1.]])
```

```
In [... U.shape
```

```
Out[... (24, 24)
```

Normalizamos la matriz de transición

```
In [... U = U / len(U)
```

```
In [... U[:1]
```

```
Out[... array([[0.04166667, 0.04166667, 0.04166667, 0.04166667, 0.04166667,
   0.04166667, 0.04166667, 0.04166667, 0.04166667, 0.04166667,
   0.04166667, 0.04166667, 0.04166667, 0.04166667, 0.04166667,
   0.04166667, 0.04166667, 0.04166667, 0.04166667, 0.04166667,
   0.04166667, 0.04166667, 0.04166667, 0.04166667]])
```

Matriz de similitud suavizada

```
In [... factor = 0.1 #valor pequeño para omitir ceros
S_s = (1 - factor)*S + factor*U
```

```
In [... S_s[:1]
```

```
Out[...]: array([[0.68444718, 0.00416667, 0.00416667, 0.00416667, 0.00416667,
   0.00416667, 0.08911048, 0.00416667, 0.00416667, 0.08696166,
   0.05614734, 0.00416667, 0.00416667, 0.00416667, 0.00416667,
   0.00416667, 0.00416667, 0.00416667, 0.00416667, 0.00416667,
   0.00416667, 0.00416667, 0.00416667, 0.00416667, 0.00416667]])
```

Explicación

La matriz original contiene valores 0.0 con los cuales es imposible calcular promedios cuando toda una fila tenga estos valores, lo que generará errores de ejecución o desbordamientos en los resultados. En este caso, usaremos el suavizado de Laplace de Markov, agregando un porcentaje mínimo de selección de los datos con la matriz de transición normalizada, luego aplicamos la fórmula $S_{-s} = (1 - \text{factor})S + \text{factor}U$ que utiliza un factor pequeño que dividirá los resultados para este caso en un 90% del total, para usar el 10% restante que modifica los ceros posibles encontrados en cada posición de la matriz.

Grafo

Debemos crearlo por aparte

```
In [...]: # Creamos el grafo
G = nx.from_numpy_array(S_s)
scores_G = nx.pagerank(G)

In [...]: # Visualizamos el grafo
plt.figure(figsize=(10,6))
pos = nx.spring_layout(G, seed=42)

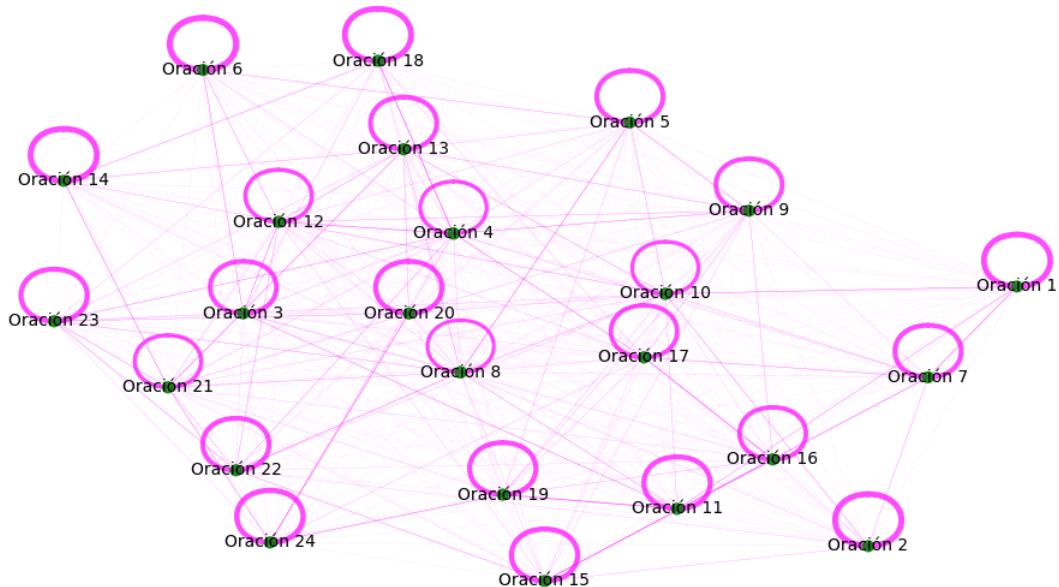
# Dibujamos Los nodos con tamaño proporcional al score_G
nx.draw_networkx_nodes(G, pos,
                       node_size=[1000 * scores_G[i] for i in G.nodes()],
                       node_color='green',
                       alpha=0.8)

# Dibujar aristas con pesos
edges = G.edges(data=True)
nx.draw_networkx_edges(G, pos,
                      width=[5 * d['weight'] for (_, _, d) in edges],
                      alpha=0.7,
                      edge_color='fuchsia')

# Etiquetas: mostrar La primera parte de cada oración
labels = {i: f'Oración {i+1}' for i in G.nodes()}
nx.draw_networkx_labels(G, pos, labels, font_size=10)

plt.title("Grafo de Similitud entre Oraciones (TextRank)")
plt.axis('off')
plt.tight_layout()
plt.show()
```

Grafo de Similitud entre Oraciones (TextRank)



Explicación

Se crea un grafo no dirigido usando `G = nx.from_numpy_array(S)`, donde cada nodo representa una oración y cada arista la similitud entre ellas. Posteriormente, se utiliza el `scores = nx.pagerank(G)` para rankear la importancia de cada oración, según como se conecta con otras.

Se crea la figura del pyplot de acuerdo al tamaño del grafo y con el `draw_networkx_nodes` se dimensionan los nodos al tamaño del ranking. Luego, el `draw_networkx_edges`, crea las aristas que interconectan cada nodo.

Finalmente, se crean las etiquetas que aparecerán en cada nodo, para nuestro caso la palabra '**Oración**' y el número del peso de la Matriz de similitud suavizada, para mostrarla a través del `plt.show()`.

TextRank

Matriz Estacionaria

Ya tenemos la matriz de transición, que nos indica la probabilidad de pasar de una oración a otra, ahora, necesitamos saber cuál es la probabilidad del estado actual en el tiempo, es decir de la oración que se está analizando sin tener en cuenta hacia qué otra oración puede ir.

```
In [...] eigen_values, eigen_vectors = np.linalg.eig(S.T)

In [...] eigen_values

Out[...] array([1.        , 0.86664673, 0.84524074, 0.82338475, 0.81472008,
   0.77307318, 0.76045412, 0.74948088, 0.73398798, 0.72032562,
   0.41731513, 0.42223906, 0.45962069, 0.46559459, 0.47809886,
   0.50087105, 0.51123721, 0.65981025, 0.56322085, 0.57564347,
   0.58091814, 0.63872781, 0.61032953, 0.62012229])
```

```
In [...] # Buscamos La posición donde el eigen_values fue 1
pos_eigen = np.where(np.isclose(eigen_values, 1.0))[0]
print(pos_eigen[0])
```

0

```
In [... # Localizamos el eigen_vector de La posición donde halló el 1.0  
eigen_vectors[:,pos_eigen[0]]]
```

```
Out[... array([-0.17088315, -0.16683293, -0.20567838, -0.2498079 , -0.19706824,  
-0.15889722, -0.19988734, -0.26248927, -0.20897602, -0.24820384,  
-0.19229987, -0.2317503 , -0.2060358 , -0.16720484, -0.18848506,  
-0.21172529, -0.21567576, -0.16973832, -0.19770387, -0.19485202,  
-0.23640176, -0.20003379, -0.19189516, -0.18299742])
```

Explicación

📌 1. ¿Qué es np.linalg.eig()? Es una función de NumPy (np) que calcula:

- Los valores propios (eigenvalues): escalares λ
- Los vectores propios (eigenvectors): vectores v

...de una matriz cuadrada A , tales que:

$$Av = \lambda v$$

📌 2. ¿Qué hace S.T? S.T es simplemente la transpuesta de la matriz S. Si S es una matriz de tamaño $m \times n$, entonces ST es de tamaño $n \times m$.

Este paso puede ser necesario si, por ejemplo, queremos que la matriz sea cuadrada para aplicar la descomposición (ya que solo matrices cuadradas tienen valores/vectores propios bien definidos).

💡 ¿Qué significan los eigenvalores y eigenvectores?

- Eigenvalor λ : Indica cuánto se estira o encoge un vector propio al aplicarle la transformación A .
- Eigenvector v : Un vector que no cambia de dirección bajo la transformación A , solo cambia su magnitud.

Puntuación

```
In [... scores = eigen_vectors[:,pos_eigen[0]]]
```

```
In [... sort_index = np.argsort(-scores)]
```

```
In [... # orden de oraciones más importantes  
sort_index]
```

```
Out[... array([ 5,  1, 13, 17,  0, 23, 14, 22, 10, 19,  4, 18,  6, 21,  2, 12,  8,  
        15, 16, 11, 20,  9,  3,  7], dtype=int64)
```

Resumen

```
In [... print('Resumen\n')  
cantidad_oraciones = 6  
for i in sort_index[:cantidad_oraciones]:  
    print("\n".join(textwrap.wrap(f'{scores[i]:.2f}: {lineas[i]}', width=50)))
```

Resumen

- 0.16: Un método muy seguro y probado aunque poco eficiente desde el punto de vista sostenible, ya que exige usar equipos computacionales cada vez más potentes que son grandes consumidores de energía
- 0.17: Esta analogía define la tarea titánica que ha supuesto para Ethereum su última actualización
- 0.17: En caso de conductas maliciosas, se podría penalizar al atacante con parte o la totalidad de los ethers que bloqueó en un inicio
- 0.17: Como el hilo de Ariadna, este ‘criptoglosario’ muestra cómo se relacionan unos con otros en la economía que viene... o que ya está aquí. Como contrapartida, preocupa la pérdida de descentralización que podría acarrear PoS
- 0.17: Como cambiarle a un coche su motor diésel por uno eléctrico mientras está en marcha
- 0.18: Un usuario puede depositar sus ‘tokens’ para convertirse en validador, participar de manera indirecta a través de un ‘exchange’, como se ha mencionado anteriormente, o prestarlos en protocolos financieros descentralizados en los que obtener también rendimientos. El debate está sobre la mesa.

Conclusiones

La tecnología del método TextRank, evidencia un avance significativo en la realización de resúmenes. Podemos observar la coherencia del texto y como se comprende fácilmente la idea principal de la noticia que se tomó como ejemplo.

Mg. Luis Felipe Bustamante Narváez

Anexo 16

Proyecto 16: Modelado de Temas con LDA

Mg. Luis Felipe Bustamante Narváez

En este ejercicio, desarrollaremos un clustering de temas de acuerdo con un dataset que contiene transcripciones de entrevistas, bastante extensas y en español. Se quiere desarrollar un clasificador que permita identificar los temas que se trabajan en cada transcripción de manera que se pueda predecir de qué habla cada una de ellas, y qué afinidad tiene con otras.

Recordemos que, **Clustering** (o agrupamiento) es una técnica de aprendizaje no supervisado cuyo objetivo es agrupar objetos similares entre sí y diferentes de los de otros grupos, sin tener etiquetas previas.

Librerías

```
In [...]
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import nltk
import textwrap
from nltk.corpus import stopwords
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.decomposition import LatentDirichletAllocation
from tqdm import tqdm
```

Manejo de StopWords

```
In [...]
nltk.download('stopwords')

[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\luis_\AppData\Roaming\nltk_data...
[nltk_data]     Package stopwords is already up-to-date!
```

Out[...]: True

```
In [...]
stop_words_original = set(stopwords.words('spanish'))
```

```
In [...]
# Convertimos el conjunto en lista
_stop_words_original = list(_stop_words_original)
_stop_words_original[:10]
```

```
Out[...]: ['sobre',
          'habíais',
          'estarían',
          'unos',
          'por',
          'estás',
          'tienen',
          'estabais',
          'eran',
          'habríamos']
```

Explicación

En una entrevista, de acuerdo con los datos que vamos a analizar, se presentan palabras adicionales a las que un texto pueda llegar a tener, palabras relacionadas con preguntas, exclamaciones, entre otras, que se repiten constantemente y que no aportan al procesamiento de datos.

Por este motivo, adicionaremos algunas palabras observadas en las entrevistas para mejorar el proceso.

```
In [...]: path_json = 'data/stop_words_news.json'
stop_words_news = pd.read_json(path_json, encoding='utf-8')
```

```
In [...]: stop_words_news[:10]
```

```
Out[...]: words
0    así
1    si
2    hacer
3    cosas
4    creo
5    cómo
6    solo
7    aquí
8    risas
9    ser
```

```
In [...]: stop_words_news = set(stop_words_news['words'])
```

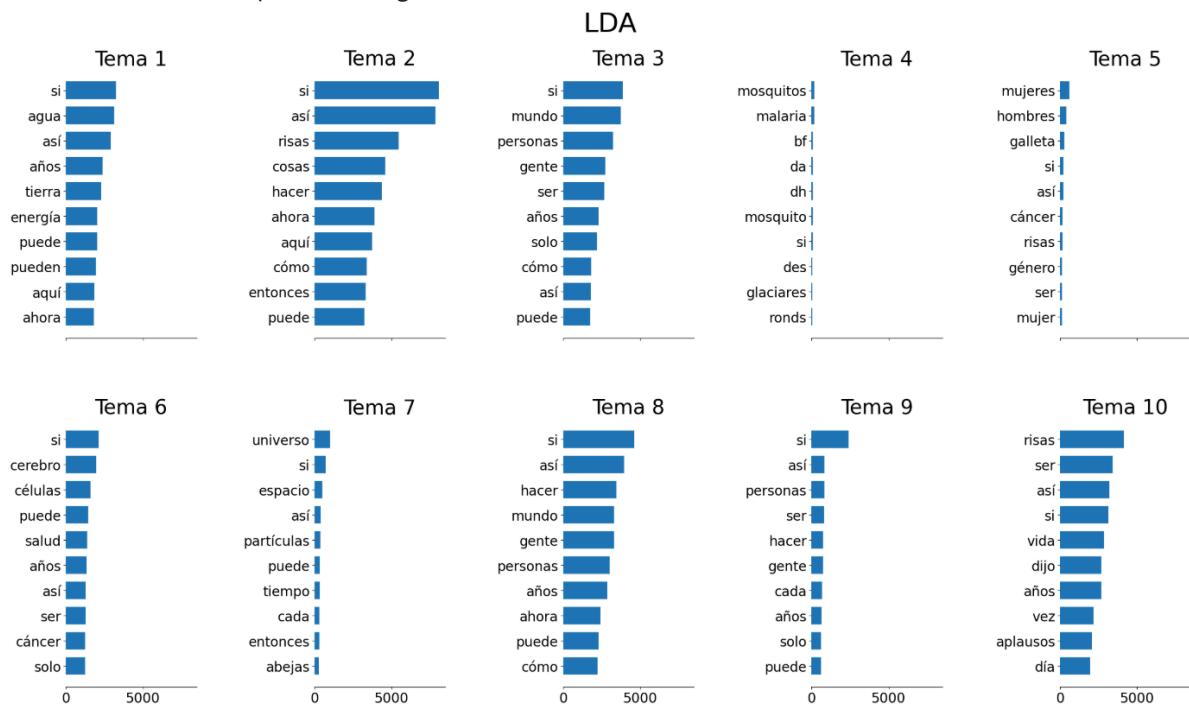
```
In [...]: stop_words_add = stop_words_original.union(stop_words_news)
```

```
In [...]: stop_words_add = list(stop_words_add)
stop_words_add[:10]
```

```
Out[...]: ['sobre',
          'habíais',
          'estarían',
          'ser',
          'sentido',
          'hombre',
          'unos',
          'fuésemos',
          'habidas',
          'por']
```

Nota: Primero realizaremos el proceso con las stopwords de la librería y luego con las que hemos añadido, de tal manera poder observar si hay diferencias en el clustering.

Resultado del LDA con las palabras originales de la librería



Posteriormente, se ejecutó con palabras añadidas; este es el producto real esperado del LDA, mostrado en las siguientes celdas de código.

Cargamos los Datos

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

```
In [...] df.iloc[:4,16:]
```

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

	url	description	transcript
0	https://www.ted.com/talks/al_gore_averting_the...	Con el mismo humor y humanidad que irradió en ...	Muchas gracias Chris. Y es en verdad un gran h...
1	https://www.ted.com/talks/david_pogue_simplifico...	El columnista del New York Times, David Pogue,...	Hola contestadora automática, mi vieja amiga. ...
2	https://www.ted.com/talks/majora_carter_greenin...	En una charla altamente emotiva, la activista ...	Si están presentes aquí hoy, y estoy muy conte...
3	https://www.ted.com/talks/sir_ken_robinson_domi...	Sir Ken Robinson plantea de manera entretenida...	Buenos días. ¿Cómo están? Ha sido increíble, ¿...

```
In [...] df['transcript'][0][:500]
```

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

'Muchas gracias Chris. Y es en verdad un gran honor tener la oportunidad de venir a este escenario por segunda vez. Estoy extremadamente agradecido. He quedado conmovido por esta conferencia, y deseo agradecer a todos ustedes sus amables comentarios acerca de lo que tenía que decir la otra noche. Y digo eso sinceramente, en parte porque – (Sollozos fingidos) – ¡lo necesito! (Risas) ¡Pónganse en mi posición! Volé en el avión vicepresidencial por ocho años. ¡Ahora tengo que quitarme mis zapatos o b'

Procesamiento de los Datos

Vectorización

```
In [... # Cada vector es una conferencia y cada conferencia tiene vectores de cada palabra
vectorizer = CountVectorizer(stop_words=_stop_words_add)

In [... # Aplicamos la vectorización a nuestro dataset
X = vectorizer.fit_transform(df['transcript'])
X

Out[... <3921x111271 sparse matrix of type '<class 'numpy.int64'>' with 1889073 stored elements in Compressed Sparse Row format>
```

Modelo LDA

```
In [... lda = LatentDirichletAllocation(
      n_components=10, #default:10
      random_state=12354, #estados aleatorios - semilla de la forma en que comienzan
    )

In [... lda.fit(X)

Out[... ▾ LatentDirichletAllocation ⓘ ?
LatentDirichletAllocation(random_state=12354)
```

Gráfico de Palabras TOP

```
In [... def graficar_palabras_top(model, feature_names, n_top_words=10):
      fig, axes = plt.subplots(2, 5, figsize=(30, 15), sharex=True)
      axes = axes.flatten()
      for topic_index, topic in enumerate(model.components_):
          top_features_ind = topic.argsort()[:-n_top_words - 1:-1]
          top_features = [feature_names[i] for i in top_features_ind]
          weights = topic[top_features_ind]

          ax = axes[topic_index]
          ax.barh(top_features, weights, height=0.7)
          ax.set_title(f'Tema {topic_index + 1}', fontdict={'fontsize': 30})
          ax.invert_yaxis()
          ax.tick_params(axis='both', which='major', labelsize= 20)
          for i in 'top right left'.split():
              ax.spines[i].set_visible(False)
          fig.suptitle('LDA', fontsize= 40)
      plt.subplots_adjust(top= 0.90, bottom=0.05, wspace=0.90, hspace=0.3)
      plt.show()
```

Explicación

La función `graficar_palabras_top` sirve para visualizar las palabras más importantes de cada tema generado por un modelo **LDA**, que es comúnmente usado en procesamiento de lenguaje natural para descubrir temas latentes en un conjunto de textos.

Primero, en la definición de la función, se reciben tres parámetros: el modelo **LDA** ya entrenado, una lista de nombres de las palabras (que suelen provenir de un **CountVectorizer** o **TfidfVectorizer**), y un número opcional que indica cuántas palabras principales por tema se quieren graficar (por defecto **10**).

Luego, se crea una figura con **10 subgráficas** organizadas en 2 filas y 5 columnas, lo cual permite visualizar hasta 10 temas. Estas subgráficas se aplanan usando **flatten()** para que sea más fácil iterar sobre ellas.

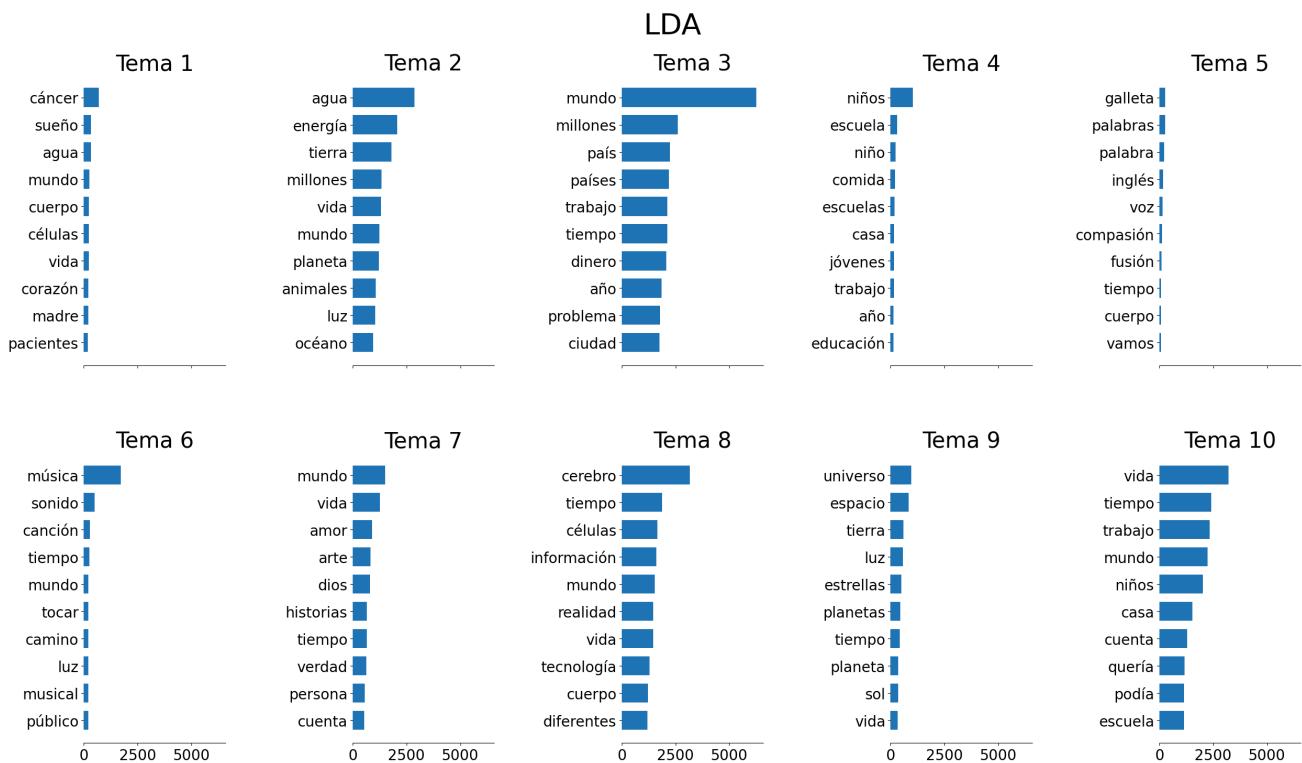
Después, comienza un ciclo con **enumerate** que recorre cada tema del modelo. Cada tema se representa como un vector que contiene los pesos (importancia) de cada palabra. Se ordenan estos pesos con **argsort()** para seleccionar las palabras más relevantes del tema, tomando sus índices. Luego, se usan esos índices para obtener tanto los nombres de las palabras como sus respectivos pesos.

Con esa información, se crea un gráfico de barras horizontal usando **barh()** en la subgráfica correspondiente, donde las palabras están en el eje Y y sus pesos en el eje X. Se le da un título a cada gráfico con el número del tema usando **set_title**, se invierte el eje Y con **invert_yaxis()** para que la palabra más importante esté arriba, y se ajustan los tamaños de fuente con **tick_params** para que sea más legible. También se eliminan algunos bordes del gráfico con **set_visible(False)** para que se vea más limpio.

Finalmente, se añade un título general a toda la figura con **suptitle()**, se ajusta el espacio entre los gráficos con **subplots_adjust()**, y se muestra el resultado con **plt.show()**.

En resumen, esta función te ayuda a interpretar de manera visual qué palabras definen cada tema extraído por el modelo LDA, lo cual es clave para entender los patrones latentes en un conjunto de textos.

```
In [...]
# Obtenemos Las palabras
palabras = vectorizer.get_feature_names_out()
# Llamamos La función para graficar
graficar_palabras_top(lda, palabras);
```



Prueba del Modelo LDA

Seleccionamos un tema al azar de toda la base de datos, y el modelo debe indicarnos de qué tema está hablando.

Matriz transformada

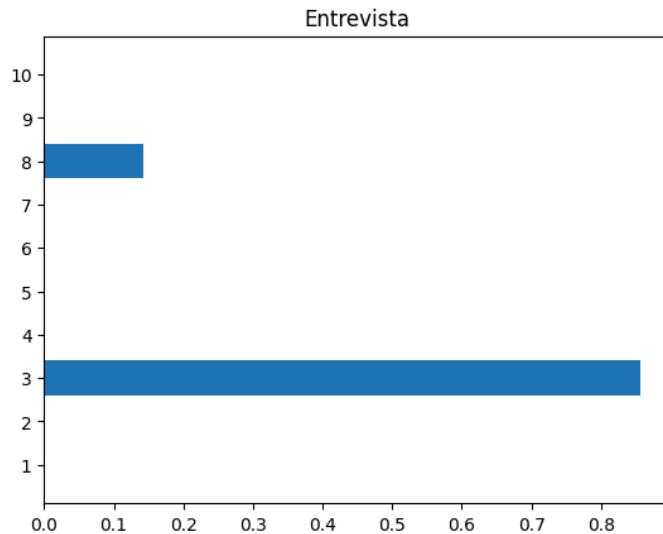
```
In [...] Z = lda.transform(X)
```

Solución gráfica

```
In [...] np.random.seed(1111) # semilla fija
i = np.random.choice(len(df))
z = Z[i]
topics = np.arange(10) + 1

fig, ax = plt.subplots()
ax.barh(topics, z)
ax.set_yticks(topics)
ax.set_title('Entrevista');
print(f'Entrevista seleccionada al azar, número {i}')
```

Entrevista seleccionada al azar, número 2460



Tema de la entrevista seleccionada - Comparación

```
In [...] def wrap(x):
    entrevista = textwrap.fill(x,
                                replace_whitespace=False,
                                fix_sentence_endings=True)
    return entrevista
```

```
In [...] print(wrap(df.iloc[i]['transcript'][:500]))
```

Hoy voy a hablar de tecnología y de la sociedad. El Departamento de Transporte estimó que, el año pasado, hubo 35 000 muertos en accidentes de auto, tan solo en EE.UU. A nivel mundial, mueren 1,2 millones de personas por año en accidentes de auto. Si hubiera una manera de eliminar el 90 % de esos accidentes, ¿apoyarían la causa? Por supuesto que lo harían. Esto es lo que promete la tecnología de vehículos autónomos, al eliminar la principal causa de accidentes: el error humano. Imagínense en el

Conclusiones

LDA, permite crear un clustering de temas y palabras asociadas a la información que trae el texto por defecto, generando un proceso estadístico capaz de filtrar la información para ofrecer mejores interpretaciones en el procesamiento de documentos.

Mg. Luis Felipe Bustamante Narváez

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

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:

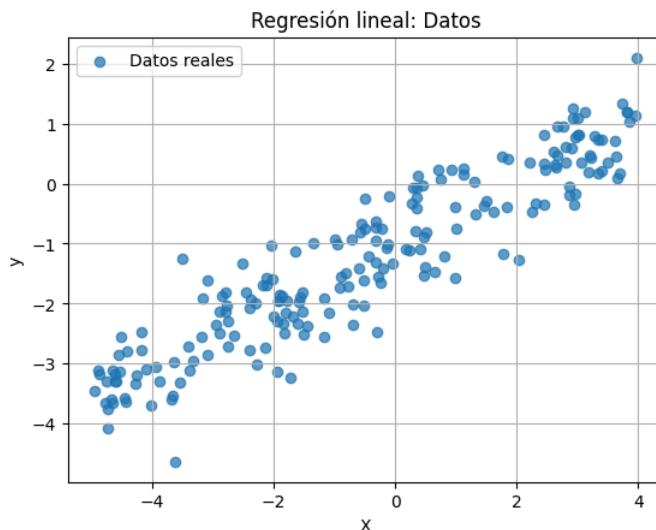
- **0.5 * X - 1** es una función lineal: $y = 0.5x - 1$.
- **np.random.randn(N)** genera 200 valores con distribución normal (media 0, desviación estándar 1).
- **0.5** ajusta el nivel del ruido: lo reduce para que tenga una desviación estándar de 0.5.

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**.

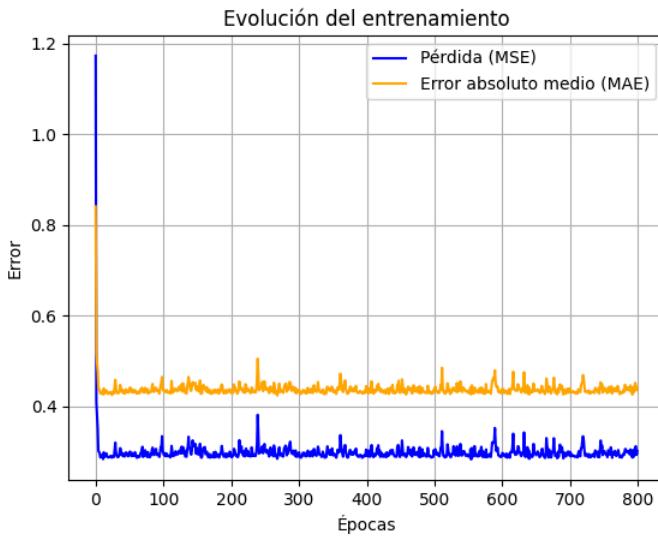
- `r.history` contiene los valores de la **función de pérdida** (`loss`) y la **métrica** (`mae`) en cada época.
 - Puedes usarlo luego para **graficar la evolución del entrenamiento**.
-

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()
```

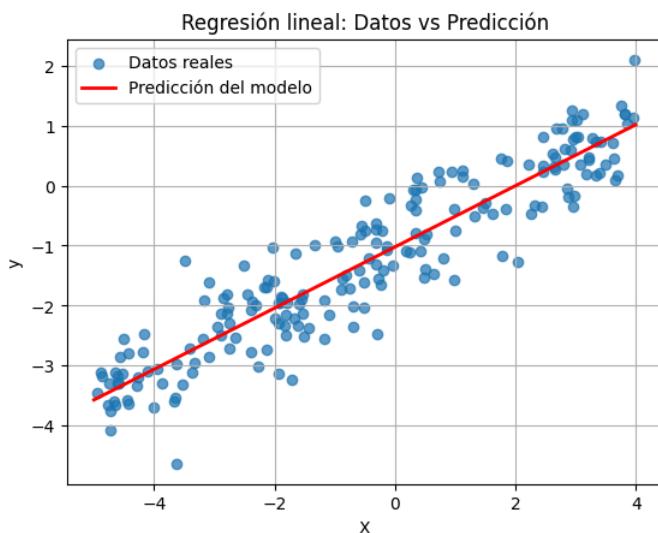


Predicción

```
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)

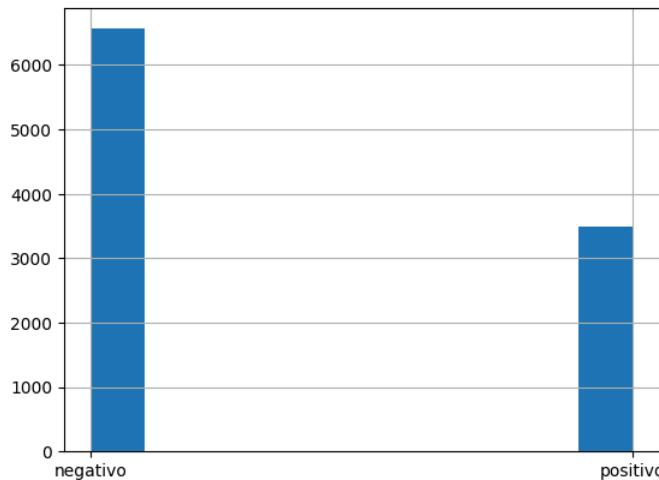
In [...]
df
```

	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
```

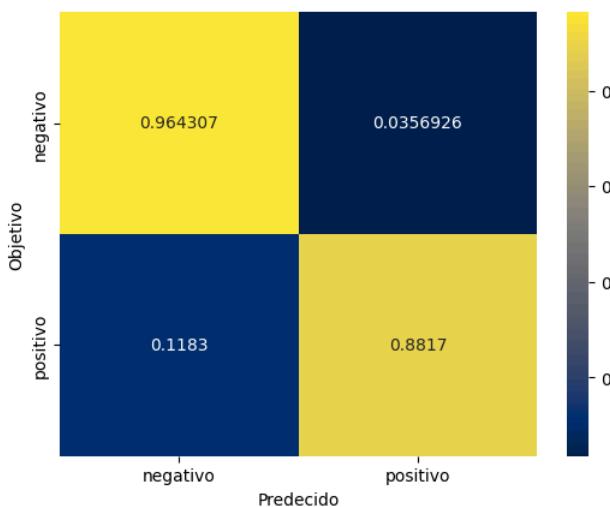
Matriz de Confusión

```
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']
```

```

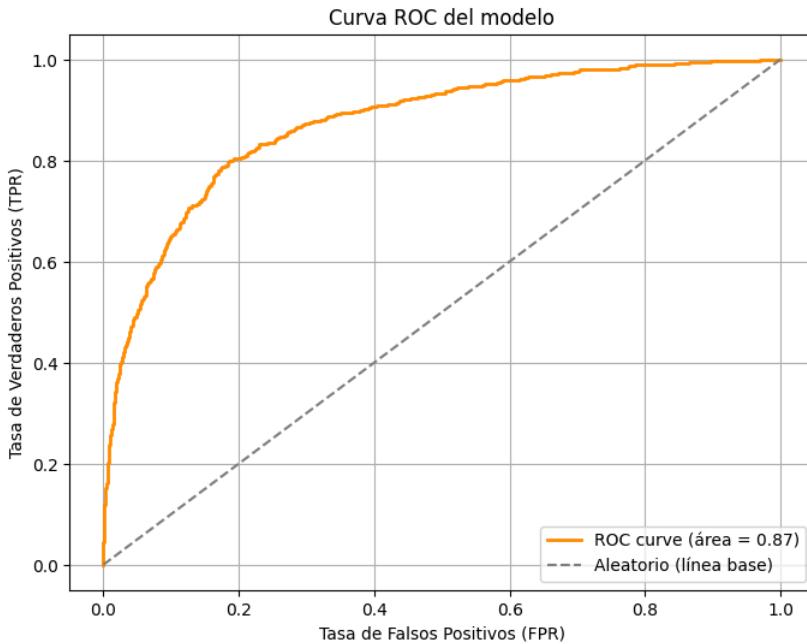
In [...] # Aplicar sigmoid a Logits de test
logits_test = modelo.predict(X_test)
probs_test = 1 / (1 + np.exp(-logits_test)) # sigmoid

# 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 ≈ 0.5 \rightarrow modelo no mejor que el azar.
 - AUC ≈ 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?

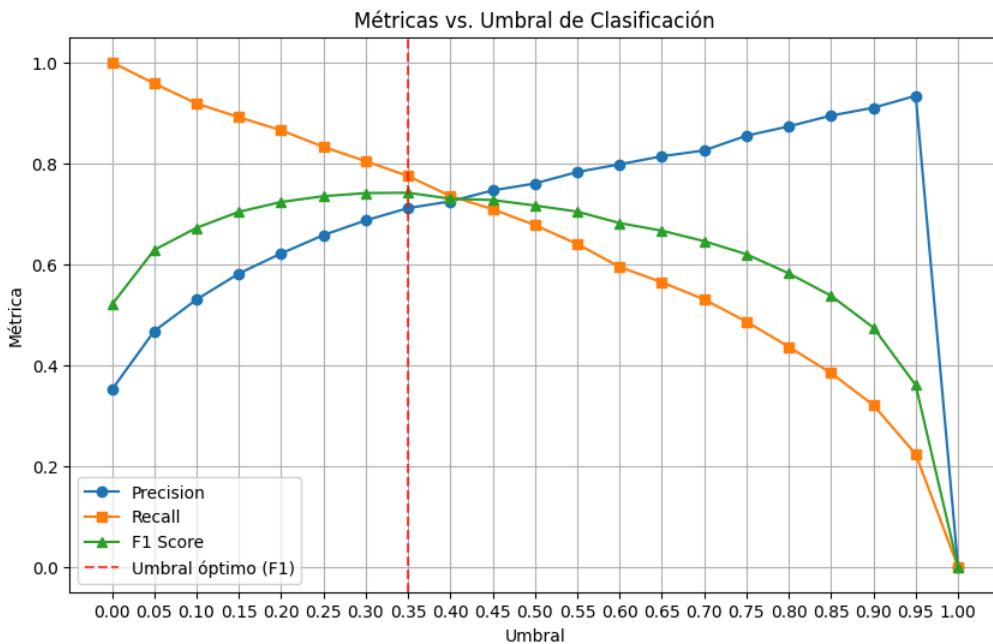
Este procedimiento permite **elegir el umbral óptimo** para convertir las probabilidades a clases, no simplemente usar el típico `0.5`. Así puedes maximizar la métrica que más te interese (en este caso el **F1 Score**, que es útil cuando hay **desequilibrio de clases**).

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

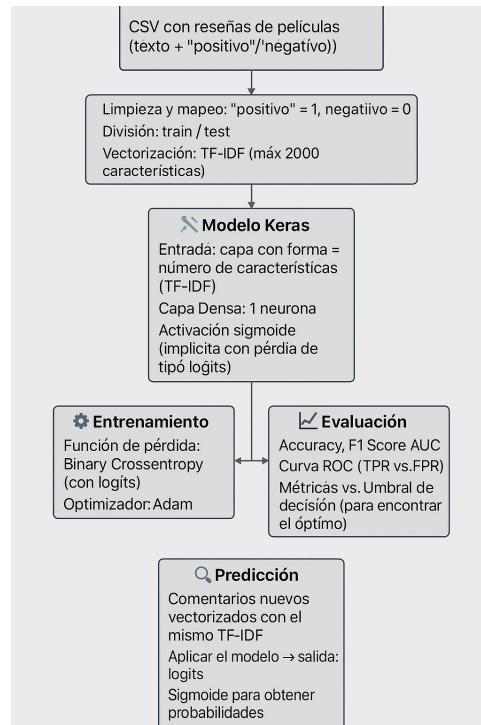
```
In [...]: prueba = [  
    'estuve muy buena la película',  
    'estuve horrible la película, me aburrió mucho',  
    'no la recomiendo'  
]  
  
# Vectorizar los nuevos textos con el mismo vectorizer usado en entrenamiento  
X_prueba = vectorizer.transform(prueba)  
  
# Realizar predicciones (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.



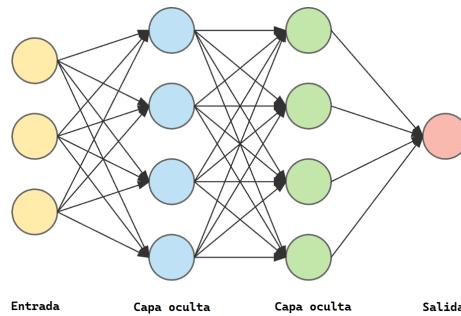
Mg. Luis Felipe Bustamante Narváez

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()
```

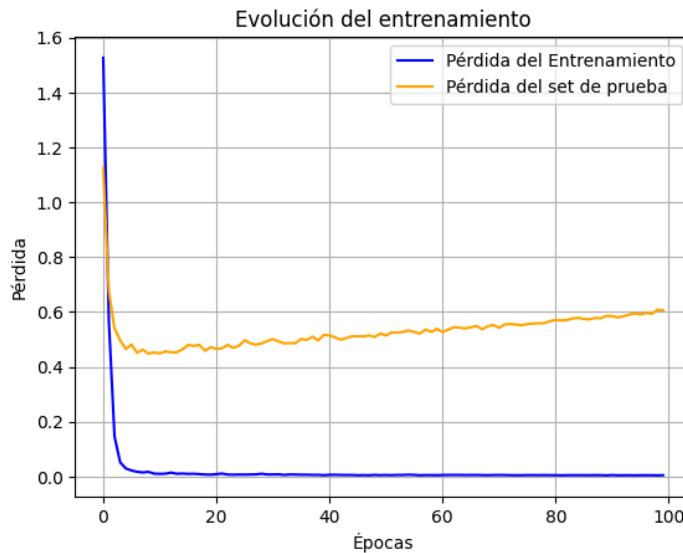
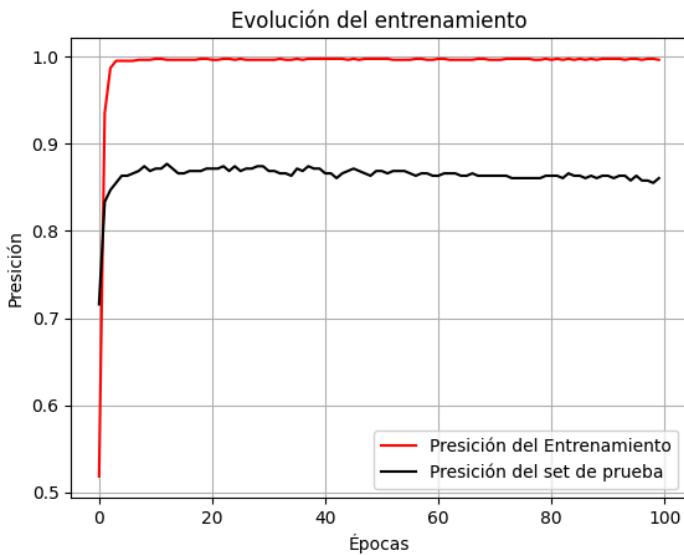


Gráfico de la presición por iteración

```
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 [...]

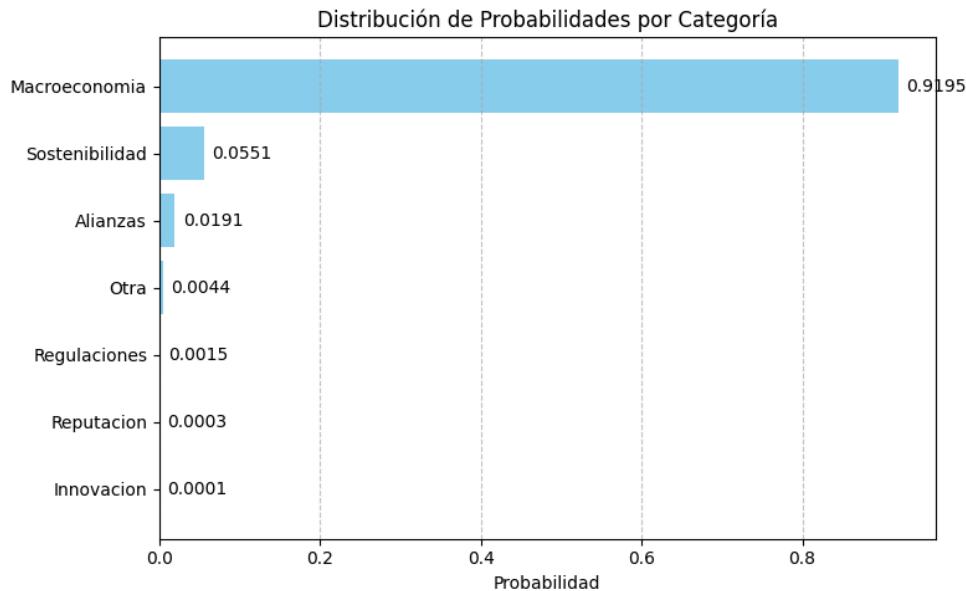
```

# 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()
```

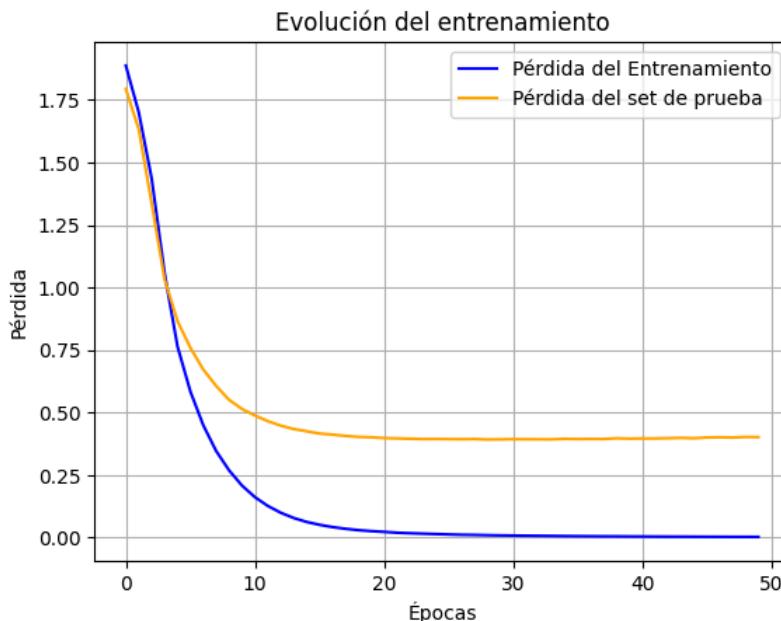
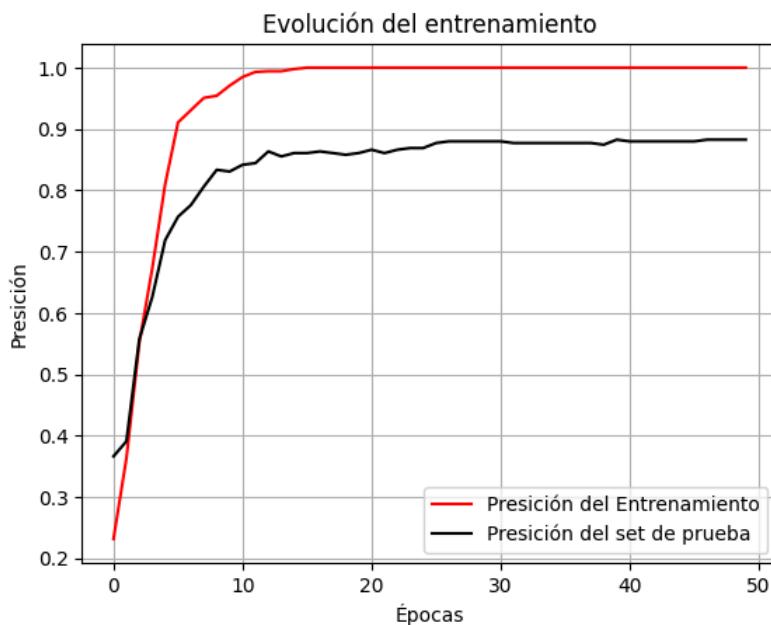


Gráfico de la presición por iteración

```
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

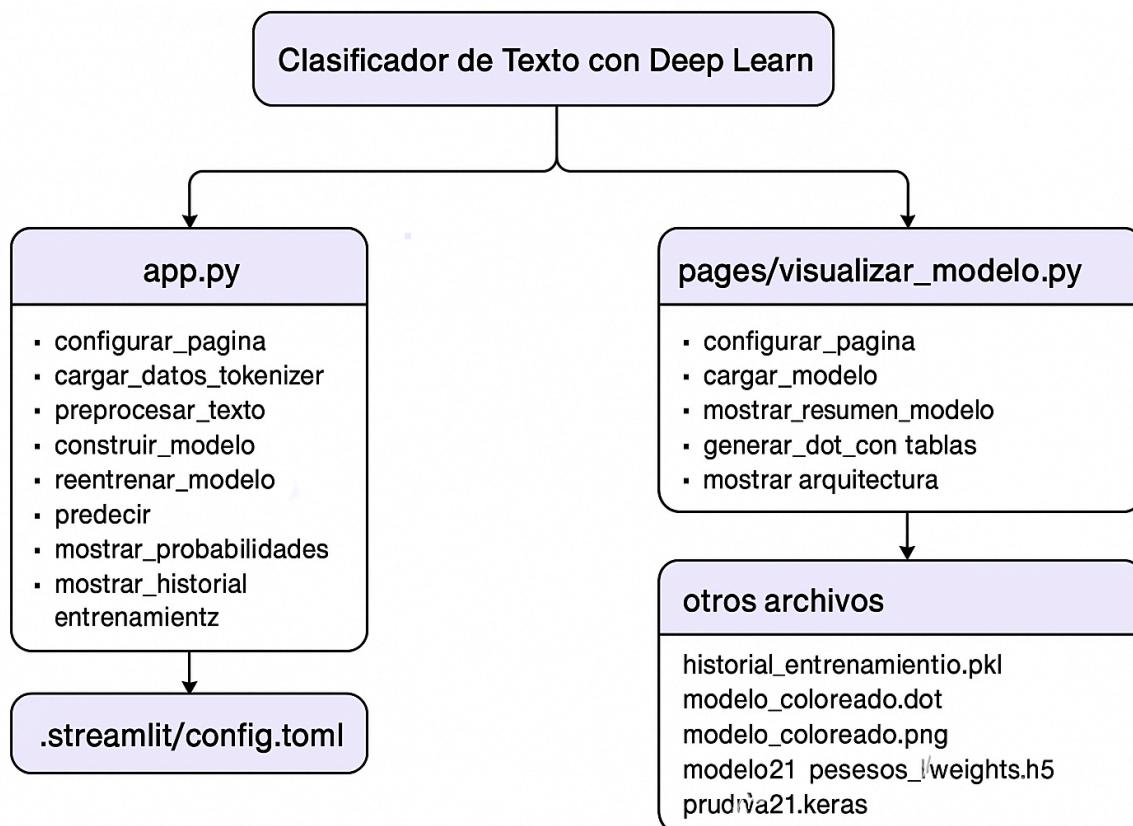
Anexo 22

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:

[!\[\]\(3564ad3a1b2e2e8b1076cccdb8bb721b_img.jpg\) **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 [...] # Añadimos 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()
```

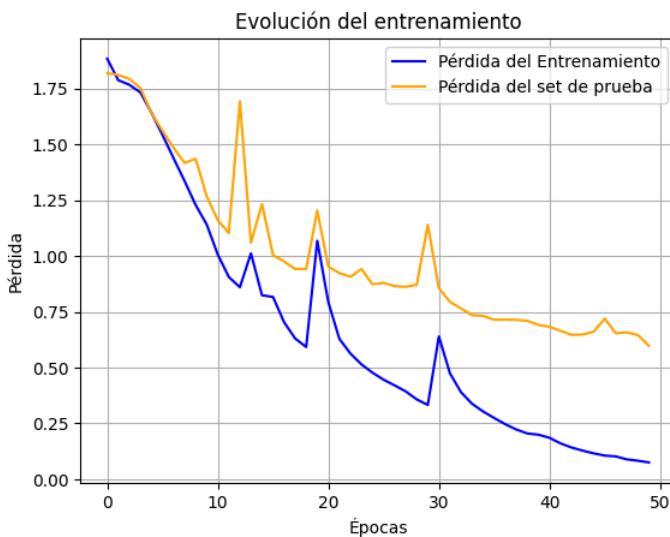
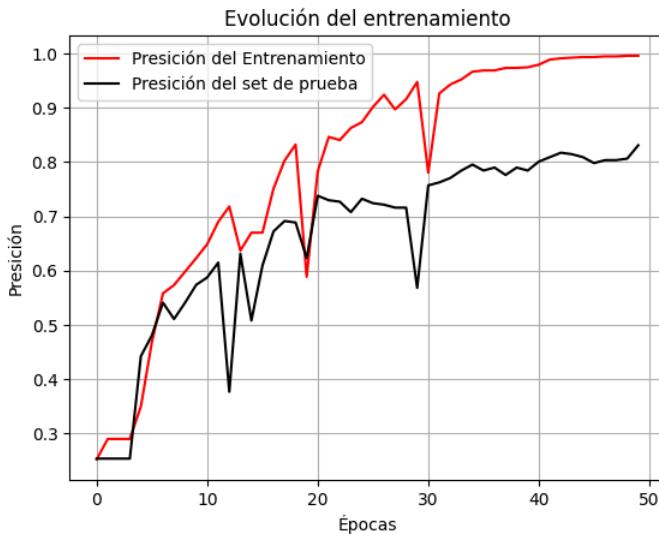


Gráfico de la presición por iteración

```
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 (depreciado) 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.

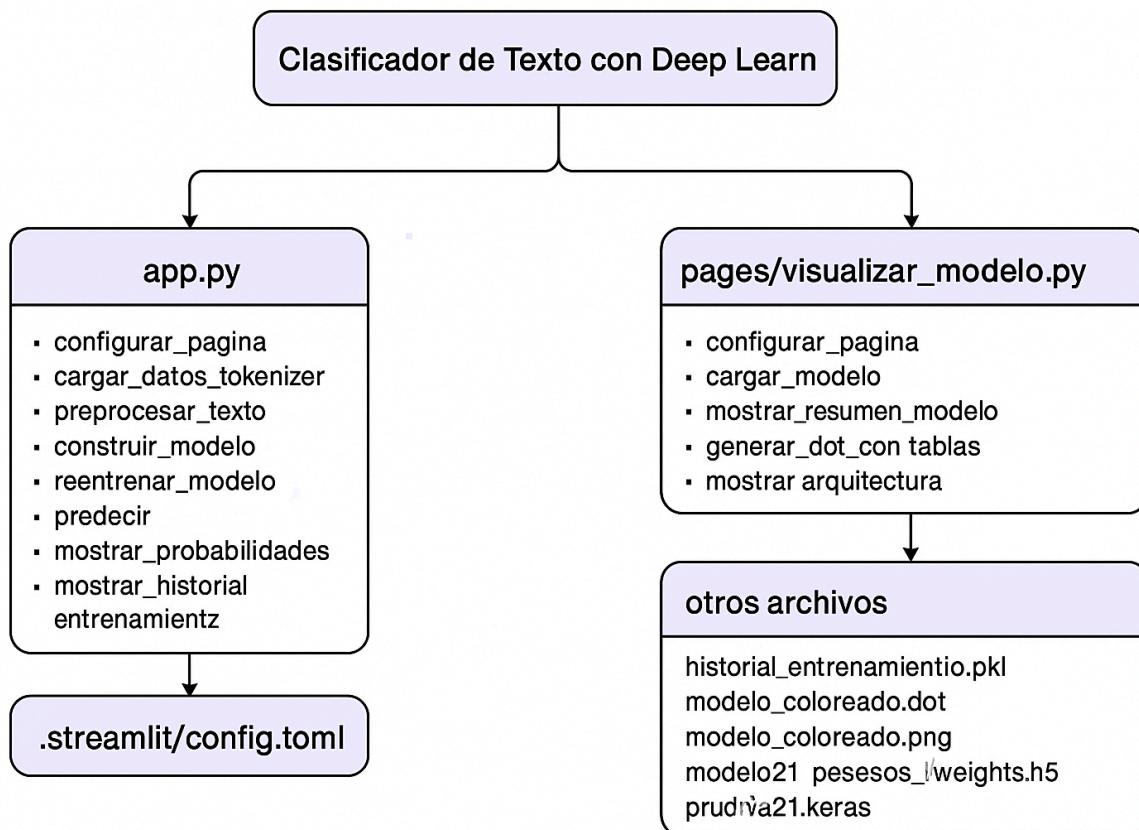
Anexo 24

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">{name} ({tipo})</TD></TR>
<TR><TD>Input</TD>
<TD>{input_shape}</TD></TR>
<TR><TD>Output</TD>
<TD>{output_shape}</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 Mg. Luis Felipe Bustamante Narváez

 GitHub ·
 LinkedIn
</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.

---

# 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):
 metricas_df = pd.read_csv(metrics_path)
 #st.info(metricas_df)
```



```

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

 if tipo == "Embedding":
 f.write(f" {node_id} --> {node_id} [label='Shape: {get_shape_safe(layer)}'];\n")
 else:
 f.write(f" {node_id} --> {node_id} [label='Shape: {get_shape_safe(layer)}'];\n\n")

 if tipo == "Conv1D" or tipo == "GlobalMaxPooling1D":
 f.write(f" {node_id} --> {node_id} [label='Weights: {layer.get_weights()}', style=dotted];\n")
 elif tipo == "Dense":
 f.write(f" {node_id} --> {node_id} [label='Weights: {layer.get_weights()}', style=dotted];\n\n")

 f.write("}")

```

```

 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">{name} ({tipo})</TD></TR>
 <TR><TD>Input</TD>
 <TD>{input_shape}</TD></TR>
 <TR><TD>Output</TD>
 <TD>{output_shape}</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>
 """)
 st.sidebar.image("images/firma.png", width=100)

```

```

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

```

```

<div class="firma-sidebar">
 Desarrollado por Mg. Luis Felipe Bustamante Narváez

 GitHub .
 LinkedIn
</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):
 metricas_df = pd.read_csv(metrics_path)
 #st.info(metricas_df)
 #st.line_chart(metricas_df[['loss', 'val_loss']])
 st.line_chart(metricas_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==3.20.0
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