FACULTAD DE CIENCIAS EXACTAS INGENIERÍA Y AGRIMENSURA

TECNICATURA UNIVERSITARIA EN INTELIGENCIA ARTIFICIAL

IA 4.4 Procesamiento del Lenguaje Natural
Trabajo Práctico N°1

Alumnos:

Ferrari, Enzo.	Legajo: F-3754/1
Pozzo, Micaela Mailen.	Legajo: P-5170/5
Rodriguez, Abril.	Legajo: R-4582/9

Profesores:

D'Alessandro, Ariel.

Geary, Alan.

Leon Cavallo, Andrea Carolina.

Manson, Juan Pablo.

2023

ÍNDICE

EJERCICIO 1

El objetivo de este ejercicio es realizar web scraping para obtener información de noticias de cuatro categorías diferentes y almacenar los datos en un DataFrame. Para cada categoría, se recopilaron los siguientes detalles de las noticias: categoría, título, texto y URL.

Se realizó un total de 10 web scraping por categoría, lo que resultó en un DataFrame llamado 'df' que contiene un total de 40 noticias con la información requerida.

Comenzamos creando dicho dataframe ‘df’ que, al principio está vacío, pero que luego de hacer webscraping por cada categoría, contendrá el resultado final del ejercicio.

```
df = pd.DataFrame(columns=["Categoria", "Titulo", "Texto", "URL"])
```

A continuación, dividimos el ejercicio por categorías, describiendo cómo fuimos trabajando para cada una de ellas.

- **Categoría ‘Economía’**

Para esta primera categoría, recolectamos las noticias de la página de Ámbito Financiero. Desde la sección contenidos, recolectamos todos los links de los artículos y luego recolectamos el texto de cada noticia.

Dentro de cada noticia, utilizamos los métodos necesarios para manejar la generación dinámica de archivos HTML.

Como la librería BeautifulSoup está hecha para manipular archivos HTML estáticos, no puede manejar contenido dinámico. Por eso, es necesario implementar otras librerías, que permitan manejar contenido JavaScript.

Para extraer datos generados por contenido generado dinámicamente, usamos la librería ‘Selenium’.

Comenzamos recolectando los links de cada noticia con el siguiente código.

```
# Recolectar link de cada noticia
url = 'https://www.ambito.com/contenidos/economia.html'

# Leer página
response = requests.get(url)
soup = BeautifulSoup(response.text, 'html.parser')

# Inspeccionando elemento sobre la página, observamos que todas las noticias
# tienen una clase de 'news-article__info-wrapper' bajo el tag div

article_tag = soup.find_all('div', class_ = 'news-article__info-wrapper')

len(article_tag)
```

```
11
```

Iteramos sobre los tags y obtenemos los links.

```
# Ahora, iteramos sobre cada uno de estos tags, observando que los links
# se encuentran bajo el href de un tag a con la clase news-article__title

links = []
for tag in article_tag:
  link = tag.select_one('.news-article__title a').get('href')
  title = tag.select_one('.news-article__title a').get_text().strip()
  links.append((title,link))
links
#tag
```

```
[('Operativos en la city: hallaron más de u$s500 mil en cajas de seguridad de cuevas vinculadas al "Croata"',
  'https://www.ambito.com/economia/allanamientos-la-city-allanan-cajas-seguridad-cuevas-vinculadas-al-croata-n5859333'),
 ('Atención empleadas domésticas: qué aumento de salario cobrarán en noviembre',
  'https://www.ambito.com/economia/atencion-empleadas-domesticas-que-aumento-salario-cobraran-noviembre-n5859780'),
 ('Chau efectivo: los pagos con medios electrónicos alcanzaron un récord',
  'https://www.ambito.com/economia/chau-efectivo-los-pagos-medios-electronicos-alcanzaron-un-record-n5859808'),
 ('Combustibles: Massa dijo que creció el abastecimiento y se normalizará en las próximas horas',
  'https://www.ambito.com/economia/massa-insiste-las-petroleras-si-no-normalizan-no-sale-un-barco-mas-exportaciones-n5859613'),
```

Si bien en la página en donde recolectamos los links no hubieron scripts de JavaScripts generando una vista dinámica, dentro de cada artículo sí. Por lo que tenemos que optar por crear una vista dinámica antes de parsear la página.

Al parecer, colab tiene problemas para crear la vista dinámica con selenium. Decidimos hacer esta parte en un entorno local, se compartirá en la entrega el proceso mediante el cual se extrajo el texto mediante un archivo .py.

Dentro del archivo .py, cargamos los links dentro de la lista ‘links’. Luego, iteramos sobre sus elementos (cada link) y realizamos webscraping.

A continuación se muestra el código que corresponde al procedimiento realizado.

```
# Iteramos sobre cada link
for title, url in links:
    # delay para no saturar la página
    time.sleep(2)
    # Inicializar una instancia WebDriver
    driver = webdriver.Chrome()
    driver.get(url)
    # Extraer el contenido dinámico de la página (primera vista)
    page_source = driver.page_source
    driver.quit()

    # Parsear el contenido con BeautifulSoup
    soup = BeautifulSoup(page_source, 'html.parser')

    # El texto se encuentra debajo de clases que comienzan con article-body
    text_tag = soup.select('[class^="article-body article-body--"]')

    # El texto está cortado, debemos juntarlo
    splitted_text = []
    for tag in text_tag:
        text = tag.get_text()
        splitted_text.append(text)

    text = ' '.join(splitted_text)

    new_row = {'title':title,
               'category':'Economía',
               'text':text,
               'url':url}
    data = data._append(new_row, ignore_index=True)
```

Finalmente, exportamos la información a un archivo csv para seguir trabajando en colab.

```
# Exportamos a un archivo csv (sería más lógico un json ¿no?)
data.to_csv('economia.csv')
```

En colab, importamos el csv que trabajamos en el .py, creamos el dataframe ‘df_economia’, y lo concatenamos al dataframe final ‘df’.

```
# Hago esto porque no quiere desaparecer Unnamed: 0
df_economia = pd.read_csv('economia.csv')[['title','category','text','url']]
df_economia.columns

# Renombramos las columnas así usamos todos los mismos nombres
df_economia.columns = ['Titulo','Categoria','Texto', 'URL']

# Concatenar al dataframe final
df = pd.concat([df, df_economia], ignore_index=True)
```

- **Categoría ‘Deportes’**

Comenzamos creando un DataFrame vacío 'df_deportes' donde se almacenará toda la información referida a esta categoría: categoría, título, texto y URL.

La página web elegida para realizar webscraping de las 10 noticias es: "https://www.cosasdedeportes.es/"

```
df_deportes = pd.DataFrame(columns=["Categoria", "Titulo", "Texto", "URL"])
```

Luego, definimos una función 'webscraping_dep' que recibe una url que corresponde a cada una de las 10 noticias encontradas y el dataframe de deportes, que luego agrega un registro (noticia) por cada iteración. Esta función realiza webscraping de cada url recibida por parámetro y devuelve el dataframe con la nueva información que respecta a dicha url.

```
def webscraping_dep(url, df):
    '''Recibe una url y un dataframe y realiza un webscraping
      de la página. Luego, almacena los datos en el df'''

    response = requests.get(url)
    soup = BeautifulSoup(response.text, 'html.parser')

    # Extraemos el título
    title = soup.find('div', {'class': 'capa_cab_post'}).find('h1').text

    # Extraemos el texto
    texto_div = soup.find('div', {'class': 'entry-content'})
    paragraphs = texto_div.find_all('p')
    texto = '\n'.join(par.text for par in paragraphs)

    # Nombre de la categoría
    categoria = "Deportes"

    # Creamos un diccionario con la info, lo convertimos en un DataFrame y lo concatenamos con el de deportes
    data = {"Categoria": [categoria],
        "Titulo": [title],
        "Texto": [texto],
        "URL": [url]}

    temp_df = pd.DataFrame(data)

    df = pd.concat([df, temp_df], ignore_index=True)

    return df
```

Luego, dentro de la variable 'url' guardamos la página de donde se extraerán los 10 blogs de la categoría Deportes. Cada blog está identificado con un id dentro de la página principal de 'cosasdedeportes.es'. A continuación, identificamos los 10 primeros id's.

Iteramos con un 'for' por cada artículo identificado con su id, y realizamos webscraping llamando a la función definida anteriormente.

```
# Obtenemos los primeros 10 ids que corresponden a cada noticia de la página de deportes
url = "https://www.cosasdedeportes.es/"
response = requests.get(url)
soup = BeautifulSoup(response.text, 'html.parser')

# Encuentramos la sección <div> con id="loop"
loop_div = soup.find('div', id='loop')

# Primeros 10 elementos <article> con atributo id
articles_with_id = loop_div.find_all('article', id=True)[:10]

# En un for, hacemos webscraping de cada noticia
for article in articles_with_id:
    link = article.find('a', class_='entry-thumbnail')
    if link:
        url = link.get('href')  # Extraigo la url de cada blog
        df_deportes = webscraping_dep(url, df_deportes)
```

Finalmente, concatenamos el dataframe con las noticias de deporte y el dataframe original.

- **Categoría ‘Inteligencia Artificial’**

Para esta tercera categoría, decidimos realizar webscraping de las 10 primeras noticias publicadas por la página "https://marketba.tech/categoria/6/inteligencia-artificial".

Comenzamos definiendo una funcion ‘webscraping_ia’. Esta función toma una URL, realiza webscraping y almacena su información en un Dataframe 'df'. Finalmente, devuelve dicho DataFrame.

```
def webscraping_ia(url, df):
    '''Recibe una url y el dataframe y realiza un webscraping
      de la página. Luego, almacena los datos en un df'''

    response = requests.get(url)
    soup = BeautifulSoup(response.text, 'html.parser')

    # Extraigo el título
    title = soup.find('div', {'class': 'fullpost__encabezado'}).find('h1').text

    # Extraigo el texto
    texto_div = soup.find('div', {'class': 'fullpost__cuerpo'})
    paragraphs = texto_div.find_all('p')
    texto = '\n'.join(par.text for par in paragraphs)

    # Nombre de la categoría
    categoria = "Inteligencia Artificial"

    # Creo un diccionario con la info, lo convierto en un DataFrame y lo concateno con el principal
    data = {"Categoria": [categoria],
        "Titulo": [title],
        "Texto": [texto],
        "URL": [url]}

    temp_df = pd.DataFrame(data)

    df = pd.concat([df, temp_df], ignore_index=True)

    return df
```

En el siguiente código, buscamos y recopilamos las URLs de las primeras 10 noticias relacionadas con inteligencia artificial en la página elegida. Las URLs se almacenan en la lista urls, lo que permite acceder a las páginas de estas noticias para realizar acciones adicionales, como el web scraping de los contenidos.

```
# Obtengo los primeros 10 ids que corresponden a cada noticia de la página de inteligencia artificial
url = "https://marketba.tech/categoria/6/inteligencia-artificial"
response = requests.get(url)
soup = BeautifulSoup(response.text, 'html.parser')

# Encuentro la sección <div> con id="loop"
loop_div = soup.find('div', class_='lista-contenido')

# Encontrar las 10 primeras noticias
noticias = soup.find_all('article', class_='post post__noticia')[:10]

urls = [noticia.find('a')['href'] for noticia in noticias]

# Creo dataframe donde se almacenarán las noticias
df_ia = pd.DataFrame(columns=["Categoria", "Titulo", "Texto", "URL"])

# Iterar a través de las URLs y realizar el scraping
for article in urls:
    df_ia = webscraping_ia("https://marketba.tech/" + article, df_ia)
```

Luego de ejecutar el código anterior, el dataFrame 'df_ia' contiene los datos extraídos de las noticias relacionadas con inteligencia artificial en las primeras 10 URLs de la página. Cada fila representa una noticia con su categoría, título, texto y URL.

Finalmente, concatenamos el df resultante (‘df_ia’) con el principal (‘df’).

- **Categoría ‘Salud’**

Para esta última categoría, decidimos realizar webscraping sobre las 10 primeras noticias de la página "https://riberasalud.com/blog-de-salud/entradas/".

Al igual que para las categorías anteriores, creamos un dataframe temporal ‘df_salud’ donde se almacenará la información luego de realizar webscrapring sobre dichas noticias.

```python
def webscraping_salud(url, df):
    '''Recibe una url y un dataframe y realiza un webscraping
      de la página. Luego, almacena los datos en el df'''

    response = requests.get(url)
    soup = BeautifulSoup(response.text, 'html.parser')

    # Extracción del título
    title = soup.find('h1', itemprop='headline').text

    # Extracción del texto
    texto_div = soup.find('div', {'class': 'entry-content'})
    paragraphs = texto_div.find_all('p')
    texto = '\n'.join(par.text for par in paragraphs)

    # Nombre de la categoría
    categoria = "Salud"

    # Diccionario con la info, lo convierto en un DataFrame y lo concateno con el principal
    data = {"Categoria": [categoria],
        "Titulo": [title],
        "Texto": [texto],
        "URL": [url]}

    temp_df = pd.DataFrame(data)

    df = pd.concat([df, temp_df], ignore_index=True)

    return df
```

Luego, hacemos una solitud HTTP a la url y obtenemos las 10 primeras noticias. Llamamos iterativamente a la función definida anteriormente, para hacer webscrapping en cada noticia y almacenar la información en el dataframe referido a la salud.

```python
url = "https://riberasalud.com/blog-de-salud/entradas/"

response = requests.get(url)
soup = BeautifulSoup(response.text, 'html.parser')

# Encuentra la sección <ul> donde se encuentran las noticias
article_list = soup.find('ul', class_='is-flex-container')

articles = article_list.find_all('li')[:10]

for article in articles:
    link = article.find('a', href=True)  # Encuentra el enlace <a> con el atributo href
    if link:
        url = link.get('href')  # Extraemos la url de cada blog
        df_salud = webscraping_salud(url, df_salud)
```

Finalmente, concatenamos el nuevo dataframe con el global 'df', donde están contenidas el resto de las noticias de las demás categorías.

El resultado de este ejercicio es el dataframe ‘df’, cuyas primeras filas se muestran a continuación.

	Categoria	Titulo	Texto	URL
0	Economía	Economía sale a buscar al menos $505.000 millo...	Aunque todavía queden por delante las dos lic...	https://www.ambito.com/economia/sale-buscar-al...
1	Economía	Petróleo: cae el precio ante esfuerzos diplomá...	El precio del barril de petróleo bajó hoy en ...	https://www.ambito.com/economia/petroleo-cae-e...
2	Economía	Los salarios subieron en agosto 7,6% y quedaro...	En agosto, el Índice de salarios se increment...	https://www.ambito.com/economia/los-salarios-s...
3	Economía	Precios de la carne al alza: cortes caros vs. ...	Después del reciente aumento en el valor del ...	https://www.ambito.com/economia/precios-la-car...
4	Economía	Desde noviembre aumentan un 50% los peajes: có...	Las tarifas de los peajes en rutas nacionales...	https://www.ambito.com/economia/desde-noviembr...
5	Economía	Amplían la bonificación de tasa en créditos pa...	La Subsecretaría de Financiamiento y Regulaci...	https://www.ambito.com/economia/amplian-la-bon...
6	Economía	Créditos con tasa del 50% para MiPyMEs: qué se...	A través de la Disposición 97/2023 - publicad...	https://www.ambito.com/economia/creditos-tasa-...
7	Economía	La economía de EEUU se aceleró al 4,9% en el t...	El producto interno bruto (PIB) de Estados Un...	https://www.ambito.com/economia/la-eeuu-crecio...
8	Economía	Mercado Pago ahora paga más por dejar la plata...	La suba de tasas que impulsó el Banco Centra...	https://www.ambito.com/economia/mercado-pago-a...
9	Economía	¿Sube el precio del pan?: actualizan valor de ...	A través de la Resolución 1860/2023 de la Sec...	https://www.ambito.com/economia/sube-el-precio...
10	Deportes	9 destinos de Golf imprescindibles para tus vi...	Si eres un apasionado del golf y estás buscand...	https://www.cosasdedeportes.es/9-destinos-de-g...
11	Deportes	¿En qué consiste Triatlón?	El orden de los tres segmentos debe de ser ést...	https://www.cosasdedeportes.es/en-que-consiste...
12	Deportes	Bicicletas eléctricas, una opción sostenible y...	Sin embargo, no todos los usuarios disponen de...	https://www.cosasdedeportes.es/bicicletas-elec...
13	Deportes	¿Cuál es la diferencia entre el tenis y el pádel?	Si nunca has practicado ninguno de los dos dep...	https://www.cosasdedeportes.es/cual-es-la-dife...

Con el método info, observamos que contiene 4 columnas de tipo object y 40 registros no nulos.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 40 entries, 0 to 39
Data columns (total 4 columns):
 #   Column     Non-Null Count  Dtype 
---  ------     --------------  ----- 
 0   Categoria  40 non-null     object
 1   Titulo     40 non-null     object
 2   Texto      40 non-null     object
 3   URL        40 non-null     object
dtypes: object(4)
memory usage: 1.4+ KB
```

De esta manera, finalizamos la resolución del ejercicio 1.

EJERCICIO 2

En base a los títulos y categorías del dataset resultante en el ejercicio anterior, el propósito es desarrollar y entrenar un modelo de clasificación de noticias.

Este modelo permitirá la asignación automática de noticias a categorías específicas basándose en el título de cada una. El objetivo principal es lograr una clasificación precisa de noticias en sus respectivas categorías (Economía - Deporte - Inteligencia Artificial - Salud), lo que puede ser de utilidad para diversas aplicaciones, como la organización y etiquetado automático de contenido.

Para lograr esto, decidimos entrenar un modelo de regresión logística que prediga la clase de cada texto. Este modelo tomará como variable explicativa el texto y como variable de respuesta la categoría.

Primero, es necesario vectorizar por el hecho de que los modelos de machine learning necesitan un input numérico, ya que no entienden el texto tal y cual como está.

Como vimos en las primeras unidades de la materia, hay múltiples maneras de vectorizar texto, como One-hot encoding, Count vectorizer y TF-IDF.

Estas técnicas pueden ser buenas para algunos propósitos, sin embargo fallan en captar el contexto de una palabra y su sintaxis. Por otro lado, tenemos otros tipos de vectorización que son aquellas que utilizan redes neuronales, en particular, las técnicas de Word Embedding.

Existen modelos pre-entrenados basados en la arquitectura BERT que nos permiten vectorizar nuestro texto. Elegimos esta técnica de vectorización para nuestro modelo.

Como variable explicativa, una concatenación del título y el texto podría dar buenos resultados, sin embargo el modelo esperaría que se le brinde la información en ese formato. Por dicha razón, usaremos sólo el texto.

Comenzamos instalando la librería ‘transformers’ con pip install.

Luego, como necesitamos que nuestra variable a predecir ‘Categoría’ tome valores numéricos, creamos una nueva columna en el dataframe ‘df’ con cada categoría codificada.

Así, la categoría ‘Economía’ tomará el valor 0, ‘Deportes’ el número 1, ‘Salud’ el número 2 e ‘Inteligencia Artificial el número 3.

A continuación, ajustamos nuestros datos a un conjunto de entrenamiento y prueba.

```python
from sklearn.model_selection import train_test_split

# Ajustamos nuestros datos a un conjunto de entrenamiento y de prueba
categorias = {'Economía':0,
              'Deportes':1,
              'Salud':2,
              'Inteligencia Artificial':3}
df['Categoria_codificada'] = df['Categoria'].map(categorias)

X = df['Texto']
y = df['Categoria_codificada']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

Luego, cargamos el modelo preentenado de BERT en Español, y definimos la función ‘get_bet_embeddings’, que recibe una lista de textos y devuelve los embeddings de BERT.

```python
# Cargar el tokenizador y modelo preentrenado de BERT para español
model_name = 'dccuchile/bert-base-spanish-wwm-cased'
tokenizer = BertTokenizer.from_pretrained(model_name)
model = BertModel.from_pretrained(model_name)

def get_bert_embeddings(texts):
    """Función para obtener los embeddings de BERT para una lista de textos."""
    embeddings = []
    for text in texts:
        inputs = tokenizer(text, return_tensors='pt', truncation=True, max_length=512)
        with torch.no_grad():
            outputs = model(**inputs)
        # Usamos el embedding del token [CLS] como la representación del texto
        embeddings.append(outputs.last_hidden_state[0][0].numpy())
    return np.array(embeddings)
```

A continuación, obtenemos los embeddings para los conjuntos de entrenamiento y prueba, llamando a la función anteriormente definida.

Entrenamos el modelo de regresión logística y lo entrenamos con los embeddings.

Finalmente, evaluamos el modelo obteniendo algunas métricas.

```python
# Obtenemos los embeddings de BERT para los conjuntos de entrenamiento y prueba
X_train_vectorized = get_bert_embeddings(X_train)
X_test_vectorized = get_bert_embeddings(X_test)

# Creación y entrenamiento del modelo de Regresión Logística
modelo_LR = LogisticRegression(max_iter=1000)
modelo_LR.fit(X_train_vectorized, y_train)

# Evaluación del modelo de Regresión Logística
y_pred_LR = modelo_LR.predict(X_test_vectorized)
acc_LR = accuracy_score(y_test, y_pred_LR)
report_LR = classification_report(y_test, y_pred_LR, zero_division=1)

print("Precisión Regresión Logística:", acc_LR)
print("Reporte de clasificación Regresión Logística:\n", report_LR)
```

Las métricas obtenidas fueron las siguientes.

```
Precisión Regresión Logística: 1.0
Reporte de clasificación Regresión Logística:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         1
           1       1.00      1.00      1.00         3
           2       1.00      1.00      1.00         1
           3       1.00      1.00      1.00         3

    accuracy                           1.00         8
   macro avg       1.00      1.00      1.00         8
weighted avg       1.00      1.00      1.00         8
```

Luego, realizamos inferencia con datos propios para evaluar el modelo.

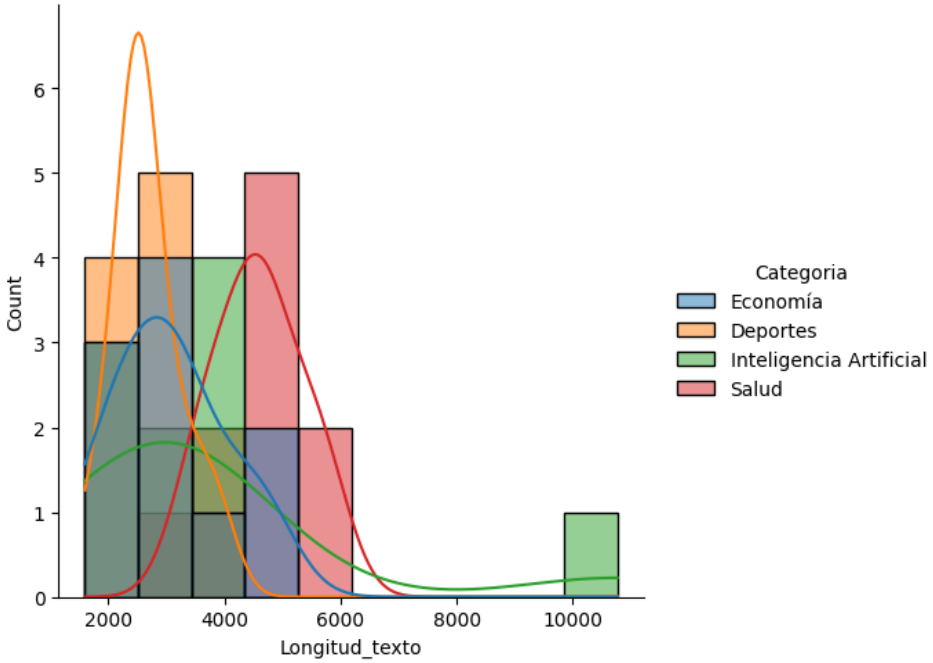
```
categorias = {'Economía':0,
              'Deportes':1,
              'Salud':2,
              'Inteligencia Artificial':3}

texto = ['Los avances recientes en IA pueden ser muy favorables para la medicina moderna',
         'La medicina moderna puede verse favorecida por la inteligencia artificial',
         'El último mes tuvo una inflación del 3%',
         'El proximo partido tendrá estrellas muy prometedoras',
         'Ella no sabía que la jeringa estaba infectada',
         'La medicina no ha traído efectos adversos hasta ahora',
         'La dieta recomendada no ha ayudado al niño, que dicho de paso, esta pasando por la adolescencia',
         'Hay que preguntarse si acaso el cáncer de mama ha podido llegar a afectar al bebe en alguna parte.']

vector = get_bert_embeddings(texto)
modelo_LR.predict(vector)
```

```
array([3, 3, 0, 1, 3, 3, 2, 2])
```

Observamos que los resultados no son todos correctos. Por ejemplo, las oraciones: 'Ella no sabía que la jeringa estaba infectada' y 'La medicina no ha traído efectos adversos hasta ahora' se categorizaron como ‘Inteligencia Artificial’ en lugar de ‘Salud’.
Con el fin de analizar estos resultados, revisamos las longitudes de los textos para así analizar si los datos están balanceados o no, lo que podría llegar a causar overfitting.



Categoria	
Deportes	26760
Economía	30564
Inteligencia Artificial	37413
Salud	45869

Analizando la distribución de cantidad de palabras en las categorías, podemos observar el balanceo de estas mismas. Notamos que no están bien balanceadas, sin embargo, el problema no es este.

Vemos que la clase Salud está underfitteada, esto quiere decir que a pesar de hacer inferencia con palabras que claramente deberían pertenecer a Salud, nos da otra categoría. Observando las noticias de esta categoría, vemos que muchas de estas no hablan directamente de salud, y es probable que categorías como IA tengan más palabras relacionadas con la salud que la misma categoría Salud.

Esto será analizado más a fondo en el próximo ejercicio, al ver la frecuencia de las palabras en cada clase.

EJERCICIO 3

El objetivo de este ejercicio es, en el contexto de cada categoría de noticias, llevar a cabo las siguientes acciones:

- Limpieza y procesamiento del texto: Se aplicarán técnicas de normalización y limpieza de texto a las noticias de cada categoría, como por ejemplo eliminar caracteres especiales, detectar y eliminar de stopwords, entre otras técnicas de preprocesamiento que consideremos necesarias.

- Nube de palabras: A partir del texto ya procesado se generará una nube de palabras que visualice de manera gráfica las palabras más relevantes en cada categoría. Esta nube de palabras proporcionará una representación visual de las palabras más importantes o que más se mencionan.

Finalmente, elaboraremos un análisis general que resuma las tendencias y características más destacadas de cada categoría en función de las palabras clave identificadas.

- **Categoría: Deportes.**

Para esta sección, hemos optado por aplicar las siguientes técnicas de preprocesamiento de texto:

- Conversión a minúsculas. Realizamos esto con el propósito de evitar que se consideren como palabras distintas aquellas que solo difieren en mayúsculas y minúsculas, por ejemplo, 'Deporte' y 'deporte'.
- Eliminación de puntuación. La puntuación, en la mayoría de los casos, no aporta información adicional o valor al análisis. Por lo tanto, decidimos eliminarla para reducir el tamaño de los datos y mejorar la eficiencia computacional.
- Eliminación de palabras de parada (stopwords). Dado que nuestro objetivo es crear una nube de palabras, resulta pertinente eliminar aquellas palabras que carecen de significado.
- Lematización. Tenemos en cuenta esta técnica con el fin de reducir las palabras a su forma base.

En cuanto a otras técnicas de preprocesamiento de texto que estudiamos en la materia, hemos decidido no aplicar algunas de ellas. Por ejemplo, dado que los 40 registros consisten en noticias o blogs redactados por profesionales o personas con habilidades de redacción, consideramos que la corrección ortográfica no es necesaria. Además, los emojis no se suelen encontrar en las noticias, por lo que no los eliminamos. La estandarización del texto tampoco la consideramos relevante en este contexto.

Comenzamos instalando es_core_news_sm y nltk con el comando pip install.

En el siguiente código, convertimos a minúsculas, eliminamos puntuación, eliminamos stopwords y lematizamos.

```
# Convertimos a minúsculas
df_deportes['Texto'] = df_deportes['Texto'].apply(lambda x: x.lower())

# Eliminamos puntuación
df_deportes['Texto'] = df_deportes['Texto'].str.replace('[^\w\s]', '')

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

def remove_stopwords(text):
    word_tokens = word_tokenize(text)
    filtered_text = [word for word in word_tokens if word.casefold() not in stop_words]
    return " ".join(filtered_text)

df_deportes['Texto'] = df_deportes['Texto'].apply(remove_stopwords)

# Lematización
text_deportes = df_deportes['Texto'].tolist()
text_completo_dep = " ".join(text_deportes)

nlp = es_core_news_sm.load()
doc = nlp(text_completo_dep)
lemmas = [tok.lemma_.lower() for tok in doc]
print(lemmas)
```

A continuación, contamos la aparición de cada palabra y la ordenamos

```
nltk.download('punkt')

text_deportes = df_deportes['Texto'].tolist()
text_completo_dep = " ".join(text_deportes)

# Tokenizamos el texto
words = word_tokenize(text_completo_dep)
sentences = sent_tokenize(text_completo_dep)

# Creamos objetos FreqDist
dic_deportes = FreqDist(words)
fdist_sentences = FreqDist(sentences)

# Imprimimos las frecuencias
dict(dic_deportes)
sorted_words = sorted(dic_deportes.items(), key=lambda x: x[1], reverse=True)
for word, frequency in sorted_words:
    print(f'{word}: {frequency}')
```

```
además: 22
si: 22
puedes: 19
cada: 14
hacer: 14
deporte: 14
```

Se observa que las palabras 'además' (22 veces), 'si' (22 veces), 'puedes' (19 veces) y 'cada' (14 veces) son las más frecuentes en los textos de las noticias de la

categoría Deportes. Sin embargo, dado que 3 de ellas no aportan significado relevante a la nube de palabras relacionada con la categoría Deportes, hemos decidido eliminarlas para evitar que aparezcan de manera prominente en la representación gráfica.

```
palabras_a_eliminar = ['además', 'si', 'cada']
palabras = word_tokenize(text_completo_dep)
# Creamos un nuevo texto sin las palabras a eliminar
text_completo_dep = ' '.join([palabra for palabra in palabras if palabra not in palabras_a_eliminar])
```

Finalmente, visualizamos la nube de palabras para esta categoría.

Vemos que, aún habiendo aplicado lematización, una de las primeras palabras que captan la atención al observar el gráfico es 'puede', que comparte el mismo significado que 'podrás' y ambas derivan del verbo 'poder'.

A pesar de esta observación, las palabras más prominentes en el gráfico, como 'deporte', 'pádel', 'mundial', 'juego', 'partido' y 'ventaja', son coherentes con el contexto de la categoría de deportes.

- **Categoría: Inteligencia Artificial.**

Para esta sección, hemos optado por aplicar las siguientes técnicas de preprocesamiento de texto:

- Conversión a minúsculas.
- Eliminación de puntuación.
- Eliminación de acentos.
- Eliminación de stopwords.
- Tratamiento de emojis. Aplicamos esta técnica porque observamos que había uno en una de las noticias.
- Conversión del texto al español. Decidimos tener esto en cuenta porque encontramos frases en inglés, las cuales son muy comunes en artículos sobre IA.
- Análisis de texto. Para esto analizamos N-gramas, co-ocurrencia de palabras y frecuencia.

Primero, convertimos el texto en minúsculas y eliminamos la puntuación, los acentos y las stopwords con el siguiente código.

```
# Conversión a minúsculas
df_ia['Texto'] = df_ia['Texto'].str.lower()

# Eliminación de puntuación
df_ia['Texto'] = df_ia['Texto'].str.replace('[^\w\s]', '')

# Eliminación de acentos
def remove_accents(input_str):
  nfkd_form = unicodedata.normalize('NFKD', input_str)
  return ''.join([c for c in nfkd_form if not unicodedata.combining(c)])

df_ia['Texto'] = df_ia['Texto'].apply(remove_accents)

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

def remove_stopwords(text):
  word_tokens = word_tokenize(text)
  filtered_text = [word for word in word_tokens if word.casefold() not in stop_words]
  return " ".join(filtered_text)

df_ia['Texto'] = df_ia['Texto'].apply(remove_stopwords)
```

Luego, como observamos un emoji en una de las noticias, decidimos tratarlo.

```
# Tratamiento de Emojis
emot_obj = emot.core.emot() # Crear una instancia de la clase emot

def replace_emoticons_and_emojis(text):
    emoji_dict = emot_obj.emoji(text)
    for emoji in emoji_dict['value']:
        text = text.replace(emoji, '')  # Reemplazar el emoji con una cadena vacía
    return text

df_ia['Texto'] = df_ia['Texto'].apply(replace_emoticons_and_emojis)
```

Como es común encontrar palabras en inglés en artículos de Inteligencia Artificial, las traducimos ejecutando el siguiente código e instalando previamente 'deep_translator' con el comando pip install.

```
from deep_translator import GoogleTranslator

# Función para traducir palabras en inglés a español
def translate_english_to_spanish(text):
    try:
        translated_text = GoogleTranslator(source='en', target='es').translate(text)
        return translated_text
    except Exception as e:
        print(f"Error al traducir: {e}")
        return text

# Aplicar la función de traducción a la columna 'Texto' en el DataFrame
df_ia['Texto'] = df_ia['Texto'].apply(translate_english_to_spanish)
```

Luego, con el fin de conocer un poco mejor los textos que intervienen en las noticias sobre IA, analizamos los N-gramas, la co-ocurrencia de palabras y finalmente su frecuencia.

Para visualizar los N-gramas, definimos una función que los genere y seteamos el valor de n-value en 2. Esto quiere decir, que el código nos devolverá bigramas.

```
def generate_and_print_ngrams(text, n):
    tokens = nltk.word_tokenize(text)
    n_grams = list(ngrams(tokens, n))
    print(f"{n}-grams para el texto:")
    for gram in n_grams:
        print(gram)

# Valor de n para los n-gramas --> 2 para bigramas
n_value = 2

for index, row in df_ia.iterrows():
    generate_and_print_ngrams(row['Texto'], n_value)
    print()
```

```
2-grams para el texto:
('presentado', 'primera')
('primera', 'vez')
('vez', 'evento')
('evento', 'inteligencia')
('inteligencia', 'artificial')
('artificial', 'compania')
('compania', '19')
('19', 'agosto')
('agosto', '2021')
('2021', 'robot')
('robot', 'proposito')
('proposito', 'general')
('general', 'promete')
('promete', 'ser')
```

Para analizar la co-ocurrencia, definimos una función que muestra las oraciones relevantes. Luego, definimos dos palabras objetivo para analizar: 'inteligencia' y 'artificial', y aplicamos la función.

```
# Función para analizar la co-ocurrencia de palabras en un texto y mostrar las oraciones relevantes
def analyze_co_occurrence(text, target_words):
    # Divide el texto en oraciones
    sentences = nltk.sent_tokenize(text)
    # Filtrar las oraciones donde co-ocurren las palabras objetivo
    co_occurrence_sentences = [sentence for sentence in sentences if all(word in sentence for word in target_words)]
    return co_occurrence_sentences

# Palabras objetivo para analizar la co-ocurrencia
target_words = ['inteligencia', 'artificial']

# Aplicar la función para analizar co-ocurrencia a la columna 'Texto' en el DataFrame
for index, row in df_ia.iterrows():
    co_occurrence_sentences = analyze_co_occurrence(row['Texto'], target_words)
    if co_occurrence_sentences:
        print(f"Co-ocurrencias de '{target_words[0]}' y '{target_words[1]}' en el texto {index}:")
        for sentence in co_occurrence_sentences:
            print(sentence)
        print()  # Agrega una línea en blanco para separar los resultados
```

Para finalizar el análisis del texto, visualizamos la frecuencia de palabras por fila con el siguiente código.

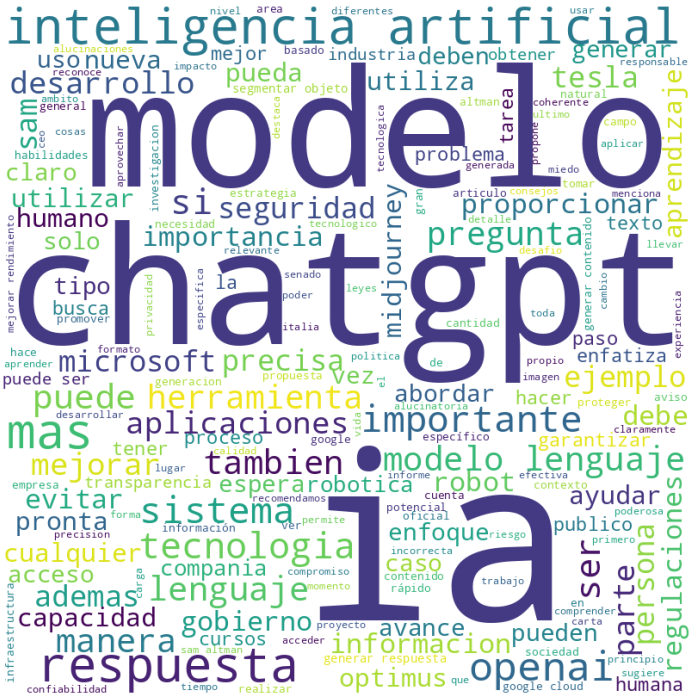
```python
# Definir una función para tokenizar y contar frecuencias
def tokenize_and_count(text):
    # Tokenizamos el texto en palabras
    words = word_tokenize(text)
    # Creamos un objeto FreqDist para las palabras
    fdist_words = FreqDist(words)
    # Devolvemos un diccionario con las palabras y sus frecuencias
    return dict(fdist_words)

# Aplicar la función a la columna 'Texto' y guardar los resultados en una nueva variable
word_frequencies = df_ia['Texto'].apply(tokenize_and_count)

# Mostrar las palabras y sus frecuencias
for index, frequencies in word_frequencies.iteritems():
    print(f"Frecuencias de palabras en fila {index}:")
    for word, frequency in frequencies.items():
        print(f"{word}: {frequency}")
    print("\n")
```

```
Frecuencias de palabras en fila 0:
presentado: 1
primera: 1
vez: 1
evento: 3
inteligencia: 4
artificial: 4
compania: 2
19: 1
agosto: 1
2021: 2
robot: 6
proposito: 1
general: 1
promete: 2
ser: 1
```

Finalmente, graficamos la nube de palabras para esta categoría.



Las palabras claves y con mayor tamaño que podemos ver en la nube de palabras dan indicaciones de la idea general del texto mostrando que el contenido de la página web de inteligencia artificial se enfoca en aspectos como modelos de IA, regulaciones gubernamentales, aplicaciones de procesamiento de lenguaje natural, chatbots GPT y la importancia de actores clave en la industria de la IA. Esta nube de palabras nos brinda un tipo de resumen del contenido del texto, y nos sirve para saber de qué se trata de una forma rápida.

- **Categoría: Salud.**

De la misma manera que antes, comenzamos convirtiendo las palabras a minúscula y eliminando puntuaciones,

```
# Convertimos a minúsculas
df_salud['Texto'] = df_salud['Texto'].apply(lambda x: x.lower())

# Eliminamos puntuación
df_salud['Texto'] = df_salud['Texto'].str.replace('[^\w\s]', '')

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

def remove_stopwords(text):
    word_tokens = word_tokenize(text)
    filtered_text = [word for word in word_tokens if word.casefold() not in stop_words]
    return " ".join(filtered_text)

df_salud['Texto'] = df_salud['Texto'].apply(remove_stopwords)

# Lematización
text_deportes = df_salud['Texto'].tolist()
text_completo_dep = " ".join(text_deportes)

nlp = es_core_news_sm.load()
doc = nlp(text_completo_dep)
lemmas = [tok.lemma_.lower() for tok in doc]
```

Por si se encuentran emojis como en el caso anterior en la categoría IA, decidimos tratarlos de la misma manera que antes.

Luego, para analizar la frecuencia de palabras en los textos, aplicamos el siguiente código y mostramos las palabras más frecuentes ordenando el diccionario (ribera, grupo, sanitario, blog).

```
text_salud = df_salud['Texto'].tolist()
text_completo_salud = " ".join(text_salud)

# Tokenizamos el texto
words = word_tokenize(text_completo_salud)
sentences = sent_tokenize(text_completo_salud)

# Creamos objetos FreqDist
dic_salud = FreqDist(words)
fdist_sentences = FreqDist(sentences)

# Imprimimos las frecuencias
dict(dic_salud)
sorted_words = sorted(dic_salud.items(), key=lambda x: x[1], reverse=True)
for word, frequency in sorted_words:
    print(f'{word}: {frequency}')
```

```
ribera: 71
grupo: 54
sanitario: 50
blog: 48
cualquier: 38
aviso: 30
contenidos: 30
comunicación: 30
salud: 28
si: 24
uso: 24
puede: 22
imágenes: 21
ningún: 20
```

Finalmente, mostramos la nube de palabras para la categoría ‘Salud’.

Se observa que todas las palabras, o al menos las primeras que uno visualiza (aquellas de mayor tamaño) están totalmente relacionadas con la categoría salud. Por ejemplo, grupo sanitario, sanitario, vitamina D, especialista, alzheimer, vida, consulta, médico, entre muchas otras.

Sin embargo, faltan palabras importantes (como por ejemplo, no se observa ninguna herramienta ni aparato médico) que pueden explicar el motivo por el cual el modelo de clasificación no dé buenos resultados al hacer inferencia sobre datos propios.

- **Categoría: Economía.**

Para esta categoría, seguimos los mismos pasos que las anteriores.

Normalizamos usando:

- Conversión a minúsculas.
- Eliminación de acentos.
- Eliminación de stopwords.
- Eliminación de puntuación.

La nube de palabras queda como la siguiente imagen.

Vemos que las palabras más frecuentes e incluso las menos frecuentes están bastante relacionadas respecto al ámbito económico. Junto con la buena predicción sobre esta categoría con el modelo de clasificación, podemos concluir que los datos representan bien a la categoría.

Conclusiones

Para las cuatro categorías se puede concluir que, luego de realizar el tratamiento de los textos, los resultados que otorgan las nubes de palabras son bastante certeros y resumen muy bien las palabras que representan a las noticias contenidas por cada categoría. Aún así, debemos contemplar la falta de palabras importantes en la categoría salud.

EJERCICIO 4

El objetivo de este ejercicio es evaluar la similitud entre los títulos de las noticias de una de las categorías mediante el uso de modelos de embedding, particularmente aquellos presentados al final de la Unidad 2. Para este caso, elegimos la categoría de ‘Inteligencia Artificial’.

Finalmente, reflexionaremos acerca de las limitaciones del modelo en base a los resultados obtenidos, en contraposición a los resultados que hubiera esperado obtener.

Realizamos un proceso de comparación de similitud entre oraciones utilizando la librería ‘sentences-transformers’.

Para ello, cargamos un modelo pre-entrenado multilingüe llamado 'distiluse-base-multilingual-cased-v1' utilizando SentenceTransformer. Este modelo se utiliza para convertir las oraciones en representaciones vectoriales.

Primero, instalamos la librería con el comando pip install. Luego, extraemos las oraciones, las codificamos y calculamos la similitud de coseno entre los vectores. El código que utilizamos es el siguiente.

```
# Cargamos el modelo preentrenado multilingüe
modelo = SentenceTransformer('distiluse-base-multilingual-cased-v1')

# Extraemos las oraciones de la columna 'Titulo' en tu DataFrame df_ia
oraciones = df_ia['Titulo'].tolist()

# Codificamos las oraciones
embeddings = modelo.encode(oraciones, convert_to_tensor=True)

# Calculamos las puntuaciones de similitud
puntuaciones_coseno = util.cos_sim(embeddings, embeddings)

# Encontramos las puntuaciones de similitud más altas
pares = []
for i in range(len(puntuaciones_coseno)-1):
    for j in range(i+1, len(puntuaciones_coseno)):
        pares.append({'index': [i, j], 'score': puntuaciones_coseno[i][j]})

# Ordenamos las puntuaciones en orden decreciente
pares = sorted(pares, key=lambda x: x['score'], reverse=True)

# Creamos una tabla para mostrar los resultados
tabla = PrettyTable()
tabla.field_names = ["Oración 1", "Oración 2", "Puntuación de Similitud"]

# Añadimos las filas a la tabla
for par in pares[0:10]:
    i, j = par['index']
    tabla.add_row([oraciones[i], oraciones[j], f"{par['score']:.4f}"])

# Mostramos la tabla
print(tabla)
```

Al final de dicho código, creamos una tabla para mostrar los resultados obtenidos. La tabla es la siguiente.

Oración 1	Oración 2	Puntuación de Similitud
Sam Altman, creador de Open IA lanza advertencia.	Sam Altman, CEO de OpenAI y su contestación curiosa sobre los planteos de expertos y Elon Musk.	0.5915
Brad Smith de Microsoft y los cinco puntos para gobernar la IA.	Sam Altman, CEO de OpenAI y su contestación curiosa sobre los planteos de expertos y Elon Musk.	0.3621
Optimus, el robot humanoide de Tesla.	Un Papa Francisco al tope de la moda con inteligencia artificial.	0.3618
Optimus, el robot humanoide de Tesla.	Sam Altman, CEO de OpenAI y su contestación curiosa sobre los planteos de expertos y Elon Musk.	0.3379
Google Cloud: Cursos gratuitos para construir tu propia inteligencia artificial.	Un Papa Francisco al tope de la moda con inteligencia artificial.	0.3173
Google Cloud: Cursos gratuitos para construir tu propia inteligencia artificial.	¿Que es un Prompt? y como mejorar tu uso de la inteligencia artificial.	0.3136
Sam Altman, creador de Open IA lanza advertencia.	SAM: un nuevo modelo de IA de Meta que segmenta cualquier objeto.	0.3114
¿Que es un Prompt? y como mejorar tu uso de la inteligencia artificial.	Un Papa Francisco al tope de la moda con inteligencia artificial.	0.3034
OpenAI aborda "alucinaciones" en ChatGPT para mejorar confiabilidad.	Sam Altman, creador de Open IA lanza advertencia.	0.2952
Optimus, el robot humanoide de Tesla.	Google Cloud: Cursos gratuitos para construir tu propia inteligencia artificial.	0.2786

Puntuación de Similitud
0.5915
0.3621
0.3618
0.3379
0.3173
0.3136
0.3114
0.3034
0.2952
0.2786

Los resultados muestran que el modelo se desempeña de manera no esperable con las expectativas, en la mayoría de las situaciones. Las similitudes son demasiado bajas.

Creemos que esto se debe a que tiene limitaciones en términos de sensibilidad al contexto, ambigüedad y complejidad de la similitud. Relaciona las oraciones entre sí y se fija si hay alguna semejanza, pero no tiene en cuenta el contexto de cada oración y la causa, consecuencia de las palabras.

Por ejemplo: en la puntuación de similitud que el modelo considera como la más baja (0.2786) las oraciones son: 'Optimus, el robot humanoide de Tesla.' y ' Google Cloud: Cursos gratuitos para construir tu propia inteligencia artificial.' En este caso el robot humanoide es construido con inteligencia artificial y ésto es una de las limitaciones que tiene el modelo, no interpreta la causa, consecuencia como relación.

En cambio, la puntuación más alta (0.5915), con las siguientes oraciones: 'Sam Altman, creador de Open IA lanza advertencia.' y 'Sam Altman, CEO de OpenAI y su contestación curiosa sobre los planteos de expertos y Elon Musk', se obtuvo porque se nombran las mismas dos palabras en ambas oraciones: 'Sam Altman'.

EJERCICIO 5

Para este ejercicio, se debe crear un programa interactivo que permita al usuario ingresar una categoría y devuelva un resumen de las noticias presentes en el dataframe.

Dado que resumir todas las noticias de una categoría resulta algo excesivo para el entorno con el cual trabajamos y los resultados no tendrán sentido alguno, decidimos hacer elegir al usuario también qué noticia quiere resumir.

La tarea de NLP que responde a las necesidades de esta aplicación es la de resumen automático de texto. Los modelos de resumen de texto se pueden dividir en: resumen extractivo (selecciona y extrae frases del texto original) y resumen abstractivo (reescribe partes del texto con nuevas frases)

Para decidir cuál de los dos enfoques usar, analizamos los pros y contras de cada uno.

- Pros de la abstracción:

- Crea un resumen más coherente, de la misma manera que lo haría un humano.
- Puede eliminar la redundancia.
- Funciona mejor para textos largos.
- Puede crear frases más fluidas, en comparación con la extracción, que solo copia frases.

- Contras de la abstracción:

- La tarea de abstracción es muy sensible a la longitud de texto resumido que tiene el conjunto de entrenamiento. Si hacemos que el modelo genere un resumen con una longitud atípicamente grande respecto al conjunto de entrenamiento, es probable que dé frases incoherentes.
- Al generar nuevo contenido, puede generar contenido incorrecto.
- Complejo.

- Pros de la extracción:

- Poco complejo.
- Mejor para textos cortos.
- Menor riesgo de generar información incorrecta o incoherente.

- Contras de la extracción:

- Redundancia, al poder extraer frases que digan lo mismo.
- Dificultad en generar frases fluidas.
- Puede no ver información que no esté presente en el texto provisto, que puede ser importante.

Al tratarse de resumir noticias, consideramos que el método de abstracción es el que mejor encaja, ya que necesitamos capturar la totalidad de la noticia.
Comenzamos definiendo una función ‘menú’, que mostrará al usuario un listado de categorías de las cuales poseemos información en el dataframe y le pedirá que ingrese una. Si el resultado no es el esperado, el programa devolverá un error y le pedirá que vuelva a ingresar un número. (“Categoría inválida. Vuelva a ingresar”)

```
def menu():
        print("\n Hola! Aquí te mostramos un listado de categorías.")
        print("1- Economía")
        print("2- Deportes")
        print("3- Inteligencia Artificial")
        print("4- Salud")
        print("5- Salir")
        print("\n")
        rta = int(input("¿Qué categoría eliges? \t"))
        while (rta != 1 and rta != 2 and rta != 3 and  rta !=4 and rta != 5):
          print("Categoría inválida. Vuelva a ingresar.")
          rta = int(input("¿Qué categoría eliges? \t"))
        return(rta)
```

Luego, buscamos modelos entrenados específicamente con noticias resumidas, para así lograr un mejor resultado. Buscando en huggingface, nos topamos con tres modelos potenciales:

- Modelo 1: Sirve para abstracción, pero está entrenado en resúmenes cortos.
- Modelo 2: Modelo de abstracción.
- Modelo 3: Si bien la carta del modelo no nos proporciona directamente con los datos de entrenamiento, al hacer pruebas vemos que logra buenos resultados con resúmenes largos, por lo que decidimos utilizar este.

```
from transformers import AutoTokenizer, AutoModel, MT5ForConditionalGeneration, BartForConditionalGeneration

modelo1 = 'LeoCordoba/mt5-small-cc-news-es-titles' # Abstracción, textos cortos
modelo2 = 'csebuetnlp/mT5_multilingual_XLSum' # Extracción
modelo3 = 'ELiRF/NASES' # Abstracción

# Cargar el tokenizador y el modelo
tokenizer = AutoTokenizer.from_pretrained(modelo3)
model = BartForConditionalGeneration.from_pretrained(modelo3)
```

Definimos una función ‘resumir’ que recibe un texto y devuelve el resumen del mismo.

```
def resumir(texto, min_length=50, max_length=100):

  # Preprocesar el texto y generar un resumen
  inputs = tokenizer.encode("summarize: " + texto, return_tensors="pt", max_length=512, truncation=True)
  summary_ids = model.generate(inputs, max_length=max_length, min_length=min_length, length_penalty=2.0, num_beams=4, early_stopping=True)
  resumen = tokenizer.decode(summary_ids[0], skip_special_tokens=True)

  # Imprimir el resumen
  return resumen
```

Finalmente, elaboramos el código principal. En él, llamamos a la función menú y guardamos la categoría elegida por el usuario en la variable ‘opcion’.
Luego, definimos una bandera, que se inicializa en 0 y toma el valor 1 cuando el usuario decide finalizar el programa eligiendo la opción 5.
A través de un bucle while y una serie de condicionales, evaluamos la selección de la categoría realizada por el usuario.

A continuación, presentamos el fragmento de código correspondiente a la opción 1 (Economía), destacando que el código principal es consistente en su estructura para todas las categorías disponibles.

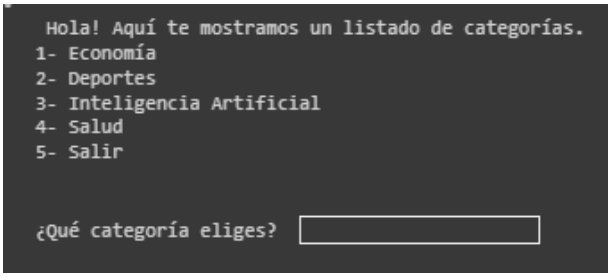
Este fragmento de código permite al usuario explorar una lista de las 10 noticias disponibles en la categoría seleccionada. Se le solicita al usuario que elija una noticia específica para obtener un breve resumen de la misma. Luego, se invoca la función ‘resumir’ y se muestra en pantalla un resumen conciso de la noticia seleccionada, junto con un enlace que proporciona acceso a la noticia completa, en caso de que el usuario desee obtener más detalles.

Finalmente, el bucle se repite, dando al usuario la oportunidad de seleccionar otra categoría o salir del programa.

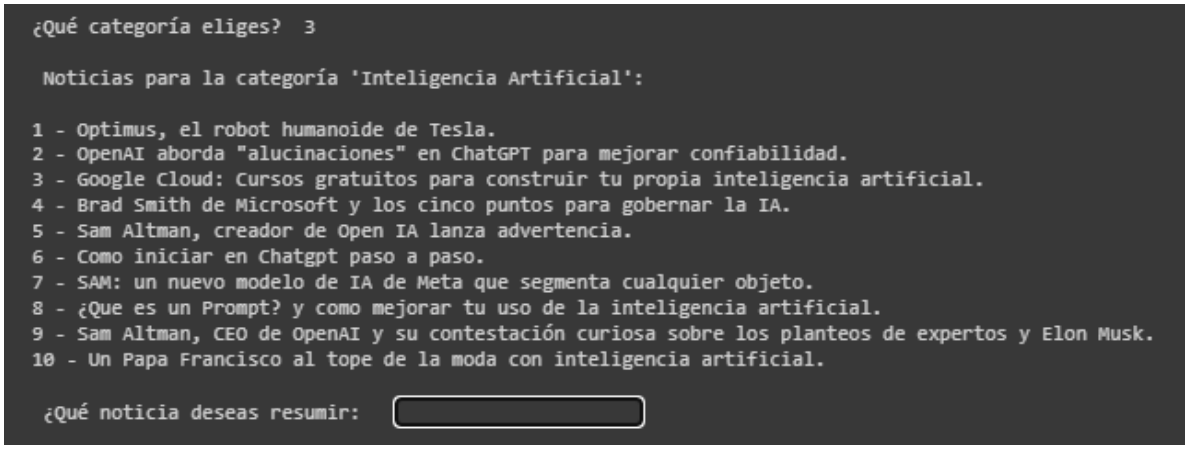
```
opcion = menu()
bandera = 0
while (bandera != 1):
 #Categoría Economía
 if opcion == 1:
   df_categoria = df[df['Categoria'] == 'Economía']
   print("\n Noticias para la categoría 'Economía': \n")
   for i, titulo in enumerate(df_categoria['Titulo']):
      print(i+1, "-", titulo)
   noticia = int(input("\n ¿Qué noticia deseas resumir: \t"))
   if 0 < noticia <= len(df_categoria):
       print("\n Aquí tienes un resumen de la noticia que has elegido: \n")
       resumen = resumir(df_categoria['Texto'].iloc[noticia - 1])
       print(resumen)
       print("\n Noticia completa: ", df_categoria['URL'].iloc[noticia - 1])
   else:
       print("Número de noticia fuera de rango")
```

A continuación mostramos el funcionamiento del programa, si se eligiera resumir una noticia de la categoría ‘Inteligencia Artificial’.

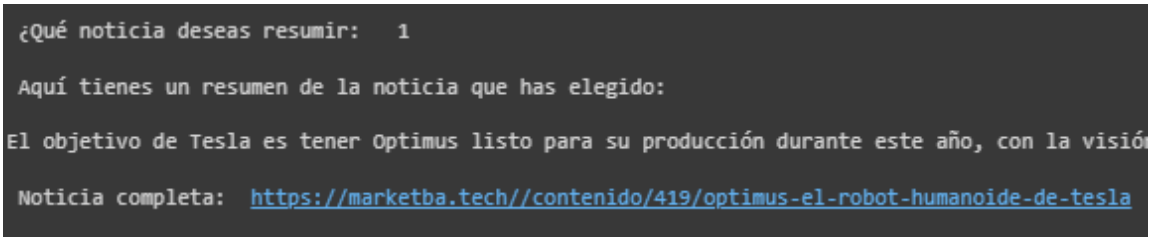
Primero, seleccionamos la categoría de interés. En este caso, la número 3.



Luego, el programa nos muestra las noticias que dispone sobre la categoría elegida. y debemos seleccionar la noticia de la cual nos interesa leer su resumen.



Elegimos una opción válida del 1 al 10 y el programa nos muestra un breve resumen de la noticia elegida. Además, nos proporciona el link de la noticia por si queremos ampliar su lectura.



BOT DE TELEGRAM

En este último ejercicio, desarrollamos un bot de Telegram utilizando la librería 'pytelegramapi'.

Este bot proporciona a los usuarios una lista de las categorías (Economía - Deportes - IA - Salud), permitiéndoles elegir un comando en función de la categoría que les interese. Una vez seleccionado el comando, el bot muestra un resumen de cinco noticias correspondientes a la categoría elegida.

Comenzamos creando un nuevo bot a través de la aplicación de Telegram utilizando 'BotFather'. Este bot se ha nombrado 'resumen_noticiasBot' y se presenta como 'Noticias resumidas por Categoría'. Al buscar su nombre de usuario en Telegram, podemos encontrarlo de manera sencilla.

En lo que respecta al código, en primer lugar, instalamos las bibliotecas 'transformers' y 'pyTelegramBotAPI' en nuestro entorno local. A continuación, importamos estas bibliotecas en un archivo .py, el cual se proporciona en la entrega del trabajo. Cargamos el archivo CSV que contiene la información relacionada con las categorías (lo exportamos desde Google Colab después de completar el ejercicio 1) y definimos una variable que almacena el token proporcionado por BotFather cuando creamos el bot.

A continuación, definimos el modelo a utilizar y una función que resume el texto que recibe, igual que en el ejercicio 5.

```python
import telebot
import pandas as pd
from transformers import AutoTokenizer, BartForConditionalGeneration

df = pd.read_csv("df.csv")
TELEGRAM_TOKEN = '6794408121:AAHn-YlpskgutxnzJBb_G8SAJWxdi3RtVw0'

modelo3 = 'ELiRF/NASES' # Abstracción
tokenizer = AutoTokenizer.from_pretrained(modelo3)
model = BartForConditionalGeneration.from_pretrained(modelo3)

def resumir(texto, min_length=50, max_length=100):
  '''Recibe un texto y genera un resumen'''
  inputs = tokenizer.encode("summarize: " + texto, return_tensors="pt", max_length=512, truncation=True)
  summary_ids = model.generate(inputs, max_length=max_length, min_length=min_length, length_penalty=2.0, num_beams=4, early
  resumen = tokenizer.decode(summary_ids[0], skip_special_tokens=True)
  return resumen
```

Inicializamos el bot con el token y creamos un mensaje de bienvenida que se muestra cuando el usuario selecciona el comando /start.

```python
# Inicializamos el bot con el token
bot = telebot.TeleBot(TELEGRAM_TOKEN)

# Mensaje cuando el usuario envía el comando /start
@bot.message_handler(commands=['start'])
def cmd_start(message):
    "Da la bienvenida al usuario y muestra las categorías disponibles"
    bot.send_message(message.chat.id,
                     "¡Bienvenido! Puedo mostrarte resúmenes de noticias de las categorías: Economía, Deportes, IA y Salud.
                     "- Para leer un resumen de noticias sobre 'Economía' escriba el comando /economia.\n\n" +
                     "- Para leer un resumen de noticias sobre 'Deportes' escriba el comando /deportes.\n\n" +
                     "- Para leer un resumen de noticias sobre 'Inteligencia Artificial' escriba el comando /ia.\n\n" +
                     "- Para leer un resumen de noticias sobre 'Salud' escriba el comando /salud.\n")
```

Además, creamos comandos específicos con la ayuda de BotFather y definimos las acciones que deben llevarse a cabo para cada uno de ellos. A continuación, se muestra el fragmento de código que se activa cuando el usuario selecciona '/economia', pero el funcionamiento es idéntico para las otras cuatro categorías.

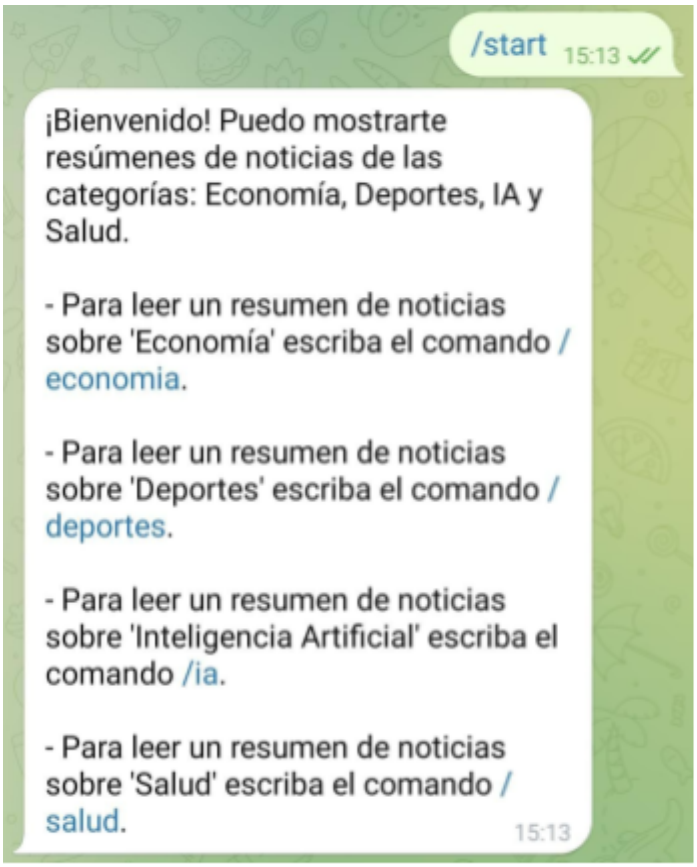
```python
@bot.message_handler(commands=['economia'])
def show_economia_news(message):
    '''Devuelve un resumen de las cinco primeras noticias
    de la categoría Economía'''
    df_categoria = df[df['Categoria'] == 'Economía']
    response = "\nNoticias sobre 'Economía':\n\n"
    for i, (titulo, texto, url) in enumerate(zip(df_categoria['Titulo'][:5], df_categoria['Texto'][:5], df_categoria['URL']
        response += f"{i+1} - {titulo}\n\n"
        resumen = resumir(texto)
        response += f"Resumen: {resumen}\n\nNoticia completa: {url}\n\n"
    bot.send_message(message.chat.id, response)
```

Por último, implementamos una validación para manejar situaciones en las que el usuario introduce un comando no válido o cualquier otro tipo de texto. También establecemos un bucle infinito que se ejecuta continuamente mientras el bot está en funcionamiento.

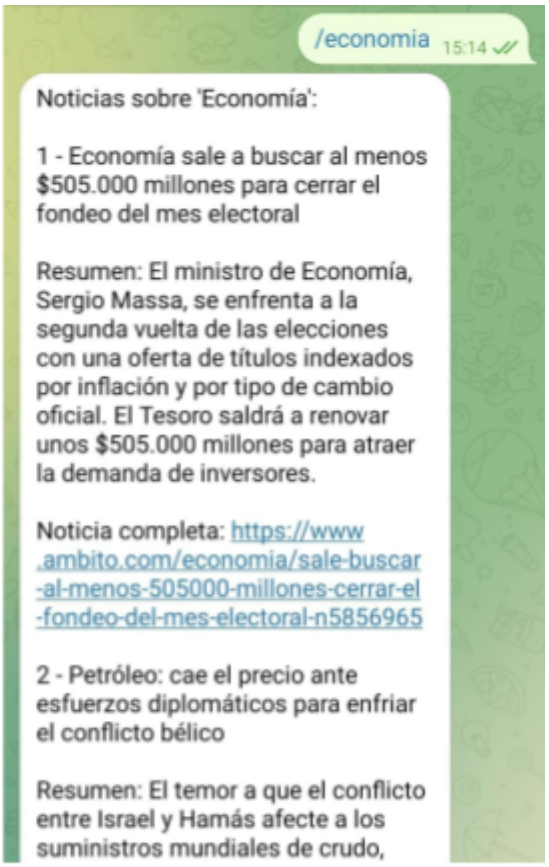
```python
@bot.message_handler(content_types=["text"])
def bot_mensajes_texto(message):
    "Gestiona mensajes recibidos no válidos"
    if message.text.startswith("/"):
        bot.send_message(message.chat.id, "Comando no disponible.")
    else:
        bot.send_message(message.chat.id, "Ingrese un comando válido. Para ver los comandos válidos ingrese /start")

if __name__ == '__main__':
    print("Iniciando bot...")
    bot.infinity_polling()
```

A continuación, mostraremos capturas de pantalla del bot en funcionamiento.



Respuesta del bot al enviar /start



Respuesta del bot al enviar una categoría.

Respuestas del bot al enviar mensajes o comandos inválidos

CONCLUSIONES

En el presente informe se detallaron los pasos y los conceptos aplicados para la resolución del Trabajo Práctico N°1 de la asignatura, cuyo objetivo fue integrar los conocimientos adquiridos en las unidades 1, 2 y 3 en cinco ejercicios.

Logramos construir un dataframe haciendo web scraping de cuatro páginas web distintas. En este conjunto de datos, construimos las columnas ‘categoría’, ‘url’, ‘título’ y ‘texto’ de las noticias extraídas. Como resultado, obtuvimos un dataframe llamado ‘df’ con las 40 noticias.

Luego, pudimos entrenar un modelo de clasificación de noticias con las categorías trabajadas, basándonos en los títulos contenidos en el dataframe. Para esto, entrenamos un modelo de regresión logística, previamente habiendo vectorizado el texto ya que es necesario un input numérico. Para llevar a cabo esta vectorización, utilizamos el modelo pre-entrenado BERT.

Realizamos la limpieza y normalización del texto contenido en las noticias utilizando diversas técnicas, como conversión a minúsculas, eliminación de puntuación, tratamiento de emojis, eliminación de stopwords, lematización, análisis de frecuencia de palabras, entre otras. De esta manera, logramos graficar las nubes de palabras que representan a cada categoría, las cuales destacan las palabras más importantes del texto.

Posteriormente, utilizamos un modelo de Embedding basado en Transformers, que utiliza la similitud del coseno, para ver la similitud entre los títulos pertenecientes a la misma categoría. Hemos analizado los resultados y explicado por qué no se ajustaron a nuestras expectativas.

Finalmente, creamos un programa interactivo que ofrece a los usuarios un resumen de noticias de su elección. Aquí, el desafío fue probar modelos de abstracción y extracción para determinar cuál de ellos generaba los resúmenes más efectivos.

Luego, implementamos este programa interactivo a través de un bot de telegram al cual se puede acceder libremente. Éste muestra un resumen de cinco noticias, dependiendo del comando que el usuario envíe. ¡Los invitamos a que puedan informarse con nuestro bot, al cual llamamos ‘resumen_noticiasBot’!