



Trabajo Práctico final
Procesamiento del lenguaje natural

Tecnicatura Universitaria en Inteligencia Artificial
Facultad de Ciencias Exactas, Ingeniería y Agrimensura

Universidad Nacional de Rosario

2023

Integrante: Micaela Pozzo.

Objetivo: el objetivo de este trabajo es integrar los conocimientos adquiridos a lo largo de todo el cuatrimestre de la materia, haciendo más hincapié en las unidades finales.

Ejercicio 1:

Para empezar, instalamos e importamos las librerías necesarias.

Descargamos archivos desde Google Drive: Se utiliza la biblioteca 'gdown' para descargar la carpeta vinculada en la URL de Google Drive y almacenarla localmente en una carpeta llamada 'PLN'.

Crear carpeta de destino: Se crea una carpeta local llamada 'datos_turismo' si no existe.

Todos los archivos de la carpeta 'PLN' se mueven a la nueva carpeta 'datos_turismo'. Se utiliza la biblioteca 'shutil' para realizar esta operación de movimiento.

Se imprime un mensaje indicando que la operación se ha completado con éxito.

```
[3] import gdown
import os
import shutil

[4] # Link con archivos sobre Turismo
url = 'https://drive.google.com/drive/folders/1kE_NMUDvvBtczjgiXip1nRbcofhdkZa?usp=drive_link'
# Descarga carpeta 'Historia Argentina'
gdown.download_folder(url, quiet=True, output='PLN')
# Crear la carpeta 'llamaindex_data' si no existe
carpeta_destino = 'datos_turismo'
if not os.path.exists(carpeta_destino):
    os.makedirs(carpeta_destino)
# Mover todos los archivos de 'Historia Argentina' a 'llamaindex_data'
carpeta_origen = 'PLN'
for filename in os.listdir(carpeta_origen):
    ruta_origen = os.path.join(carpeta_origen, filename)
    ruta_destino = os.path.join(carpeta_destino, filename)
    shutil.move(ruta_origen, ruta_destino)
# Eliminar la carpeta 'Historia Argentina'
shutil.rmtree(carpeta_origen)
print("Archivos movidos con éxito.")

Archivos movidos con éxito.
```

Una vez que tenemos nuestros PDFs en la carpeta, extraemos el texto.

Se define una función llamada 'extraer_texto_pdf' que toma la ruta de un documento PDF, abre el documento con PyMuPDF y extrae el texto de cada página, acumulándolo en una cadena que se devuelve al final.

Se imprime el nombre de cada documento y el texto extraído de ese documento.

Extraer texto

```
import fitz # PyMuPDF
import os

# Ruta de la carpeta que contiene los documentos PDF
carpeta_datos_turismo = '/content/datos_turismo'

# Función para extraer texto de un documento PDF
def extraer_texto_pdf(ruta_documento):
    texto = ''
    doc = fitz.open(ruta_documento)
    for pagina_num in range(doc.page_count):
        pagina = doc[pagina_num]
        texto += pagina.get_text()
    return texto

# Lista para almacenar texto extraído de cada documento
textos_documentos = []

# Iterar sobre los archivos PDF en la carpeta
for filename in os.listdir(carpeta_datos_turismo):
    if filename.endswith('.pdf'):
        ruta_documento = os.path.join(carpeta_datos_turismo, filename)
        texto_documento = extraer_texto_pdf(ruta_documento)
        textos_documentos.append({'nombre': filename, 'texto': texto_documento})

# Imprimir el texto extraído de cada documento
for documento in textos_documentos:
    print(f"Nombre del documento: {documento['nombre']}")
    print(f"Texto extraído: {documento['texto']}\n")
```

+ Código + Texto

```
ruta_documento = os.path.join(carpeta_datos_turismo, filename)
texto_documento = extraer_texto_pdf(ruta_documento)
textos_documentos.append({'nombre': filename, 'texto': texto_documento})

# Imprimir el texto extraído de cada documento
for documento in textos_documentos:
    print(f"Nombre del documento: {documento['nombre']}")
    print(f"Texto extraído: {documento['texto']}\n")
```



Nombre del documento: INTRODUCCION-AL-TURISMO-OMT.pdf
Texto extraído: Introducción
al Turismo
OMT Organización Mundial del Turismo
DIRECCIÓN
Amparo Sancho
COLABORAN
Dimitrios Buhalis
Javier Gallego
Jaume Mata
Susana Navarro
Estefanía Osorio
Aurora Pedro
Sergio Ramos
Paz Ruiz

El trabajo que se presenta no hubiese sido posible sin la iniciativa del Secretario General de la Organización Mundial del Turismo, Ilmo. Sr. D. Francisco Frangialli y de su departamento de desarrollo de recursos humanos, que ha sido capaz de detectar la ausencia de material educativo para cubrir el gran vacío que, en materia turística, existe en el campo de los contenidos básicos. Tampoco se habría llevado a cabo sin la intervención conjunta de un grupo de profesores y profesionales del sector turístico y de la persona de la que todos, de una forma u otra, hemos aprendido los conocimientos turísticos: el profesor D. Eduardo Fayos Solá. Deseo agradecer, igualmente, la ayuda recibida por todas las personas que han leído las versiones preliminares del libro y cuyos interesantes comentarios se han incorporado a esta obra: el profesor D. Bernardí Cabrer, el profesor D. Antonio Juárez, el profesor D. Francisco González-Palazón, Dña. Adela Morada, Dña. Cristina Alegría, Dña. Blanca Olivares, Dña. Ainhoa

✓ 1 min 18 s completado a las 15:59

Realizamos splitting al texto extraído, para esto usamos la librería 'langchain'

Se itera sobre la lista de documentos y para cada documento, se divide el texto en párrafos usando `split('\n')` y se imprime el nombre del documento y cada párrafo junto con su número de orden.

```
# Ruta de la carpeta que contiene los documentos PDF
carpeta_datos_turismo = '/content/datos_turismo'

# Función para extraer texto de un documento PDF
def extraer_texto_pdf(ruta_documento):
    texto = ''
    doc = fitz.open(ruta_documento)
    for pagina_num in range(doc.page_count):
        pagina = doc[pagina_num]
        texto += pagina.get_text()
    return texto

# Lista para almacenar texto extraído de cada documento
textos_documentos = []

# Iterar sobre los archivos PDF en la carpeta
for filename in os.listdir(carpeta_datos_turismo):
    if filename.endswith('.pdf'):
        ruta_documento = os.path.join(carpeta_datos_turismo, filename)
        texto_documento = extraer_texto_pdf(ruta_documento)
        textos_documentos.append({'nombre': filename, 'texto': texto_documento})

# Dividir el texto extraído en párrafos o líneas
for documento in textos_documentos:
    nombre_documento = documento['nombre']
    texto_documento = documento['texto']

    # Dividir el texto en párrafos (puedes usar '\n' para líneas)
    parrafos = texto_documento.split('\n')

    # Imprimir los párrafos
    print(f"Nombre del documento: {nombre_documento}")
    for i, parrafo in enumerate(parrafos, 1):
        print(f"Párrafo {i}: {parrafo}")
    print("\n")
```

✓ 1 min 48 s completado a las 15:50

```
Párrafo 5280: El fuerte impacto de la crisis en trabajadores
Párrafo 5281: informales, que no cuentan con acceso a
Párrafo 5282: mecanismos de protección social contributivos,
Párrafo 5283: puso en evidencia la necesidad de desarrollar
Párrafo 5284: medidas de protección social basadas en las
Párrafo 5285: transferencias. En este sentido, el gobierno pagó
Párrafo 5286: un bono extraordinario para las personas que
Párrafo 5287: cobran la Asignación Universal por Hijo (AUH)
Párrafo 5288: y para jubilados y jubiladas que perciben una
Párrafo 5289: única jubilación o pensión mínima, así como la
Párrafo 5290: postergación del pago de las cuotas para créditos
Párrafo 5291: de la Administración Nacional de la Seguridad
Párrafo 5292: Social (ANSES).
Párrafo 5293: La medida más importante fue el Ingreso
Párrafo 5294: Familiar de Emergencia (IFE), debido a su amplia
Párrafo 5295: cobertura horizontal y su llegada a aquellos
Párrafo 5296: grupos que no pueden acceder a ningún otro tipo
Párrafo 5297: de apoyo a los ingresos. Se trata de una prestación
Párrafo 5298: monetaria no contributiva de 10 000 pesos
Párrafo 5299: argentinos destinada a las personas argentinas,
Párrafo 5300: o con residencia legal en el país desde al menos
Párrafo 5301: dos años, entre 18 y 65 años de edad, que se
Párrafo 5302: encuentren desocupadas, se desempeñen en la
Párrafo 5303: economía informal, monotributistas inscriptos en
Párrafo 5304: las categorías inferiores o trabajadoras de casas
Párrafo 5305: particulares (Ernst, Mourello 2020).
Párrafo 5306: Posteriormente, la gradual salida de la etapa
Párrafo 5307: de restricciones sanitarias más severas dio
Párrafo 5308: lugar también a una gradual recuperación de
Párrafo 5309: la actividad económica. Esto llevó primero a la
Párrafo 5310: reducción del número de personas trabajadoras
Párrafo 5311: asistidas por el programa ATP y luego a su
Párrafo 5312: cancelación. Este proceso se desarrolló en la
Párrafo 5313: mayoría de las actividades económicas con la
Párrafo 5314: excepción de las actividades del turismo cuya
Párrafo 5315: recuperación se dificulta más y donde el nivel de
Párrafo 5316: cubrición estatal no cambió de manera me...
```

✓ 1 min 18 s completado a las 15:59

Creación de Embeddings:

se utiliza la biblioteca 'spaCy' para procesar los textos extraídos de documentos PDF y obtener embeddings (representaciones vectoriales) promedio de cada documento.

Se carga el modelo de spaCy para español. En este caso, se utiliza el modelo 'es_core_news_sm', que es un modelo pequeño para procesamiento en español.

Se crea una lista vacía llamada 'embeddings_documentos' que se utilizará para almacenar los embeddings de cada documento.

Se itera sobre la lista de documentos que contiene el nombre del documento y su texto extraído. Para cada documento, se utiliza spaCy para procesar el texto y se obtiene el embedding promedio de todas las palabras en el documento. El nombre del documento y su embedding se añaden a la lista 'embeddings_documentos'.

Se itera sobre la lista de embeddings y se imprime el nombre de cada documento junto con su embedding correspondiente.

Los embeddings son representaciones vectoriales que capturan la semántica y el significado de las palabras en el texto. Esto puede ser útil para realizar análisis semánticos o comparar la similitud entre diferentes documentos en función de sus contenidos.

```
import spacy

# Cargar el modelo de spaCy
nlp = spacy.load('es_core_news_sm')

# Lista para almacenar embeddings de cada documento
embeddings_documentos = []

# Iterar sobre los textos extraídos
for documento in textos_documentos:
    texto = documento['texto']
    # Procesar el texto con spaCy
    doc = nlp(texto)
    # Obtener el embedding promedio de las palabras en el documento
    embedding = doc.vector
    embeddings_documentos.append({'nombre': documento['nombre'], 'embedding': embedding})

# Imprimir los embeddings de cada documento
for documento in embeddings_documentos:
    print(f"Nombre del documento: {documento['nombre']}")
    print(f"Embedding: {documento['embedding']}\n")
```

```

Nombre del documento: INTRODUCCION-AL-TURISMO-OMT.pdf
Embedding: [ 0.40163356 -0.06478172 -0.29960343 0.07274809 0.34361285 0.32876173
0.00724786 0.24722986 -0.23534812 0.2219148 0.06537892 -0.01084483
-0.12531506 0.77016836 0.13271874 0.9242674 -0.06636347 0.2179811
-0.46226776 -0.24315652 -0.06144148 0.315668 0.3674513 0.2295553
-0.3811527 0.0869769 0.62738246 -0.37002403 -0.15908201 0.8013801
-0.12944555 -0.35028002 -0.05825097 -0.05837705 0.0835449 0.5361501
0.63658106 -0.48857826 0.42567125 -0.1886921 -0.65976083 -0.05550978
0.20244884 -0.23946792 0.3650718 -0.26764026 0.3640573 0.73704916
-0.12380952 0.61883754 -0.21958202 0.29335746 0.53375286 0.1817371
-0.12802339 -0.24468866 -0.01495889 -0.312765 -0.4292746 -0.94107807
-0.478146 -0.2814638 0.02208436 -0.03592857 -0.38894013 0.11554345
0.8298405 -0.4462649 0.45911485 -0.3497752 -0.76043534 -0.10139197
0.40839612 0.05454683 -0.9941222 0.1940461 -0.01212187 0.04114867
-0.05136645 0.07058024 0.14604932 0.05198516 0.07767324 -0.0621305
-0.5293585 0.23371863 -0.24770863 0.48324072 0.32792363 -0.08841341
0.2916187 -0.11168291 -0.43778488 0.33680737 -0.7516376 -0.05807598]

```

```

Nombre del documento: wcms_853411.pdf
Embedding: [ 0.35886645 -0.08279838 -0.45106137 0.08059218 0.44858697 0.70176977
0.07510947 0.27380806 -0.39356425 0.28716382 0.13610895 0.12577914
-0.18017176 1.1105384 0.17004178 0.7893878 -0.2508045 0.37828803
-0.5075357 -0.04787606 -0.0011635 0.43406427 0.55856526 0.29951176
-0.506703 0.15135685 0.536209 -0.49131963 -0.36213052 0.7027059
-0.22286144 -0.41255537 -0.22992113 -0.0846669 0.10037865 0.648087
0.9040539 -0.50294906 0.42806742 -0.22716133 -0.86864036 -0.12387287
0.2779796 -0.2861849 0.38226134 -0.0324792 0.2877822 0.8837616
-0.01439555 0.7374808 -0.4145338 0.36015853 0.43725076 0.14682922
-0.27321354 -0.2585926 0.14387041 -0.41840592 -0.38428247 -0.93104815
-0.5910826 -0.19869795 0.1617811 -0.15289517 -0.54745203 0.1371274
0.94154537 -0.5989138 0.11585059 -0.35563824 -0.73842055 -0.09955754
0.511274 0.05229065 -0.8549403 0.18899223 -0.02240241 -0.12991902

```

Almacenar los datos en una base de datos vectorial:

CrhomaDB

Se crea una instancia del cliente de la base de datos utilizando la clase `Client` proporcionada por la biblioteca `chromadb`. Este objeto `client` se utilizará para interactuar con la base de datos.

Se utiliza el cliente para crear una colección (similar a una tabla en bases de datos relacionales) en la base de datos. La colección se llama "my-collection". La variable `'collection'` guarda una referencia a la colección recién creada y se puede utilizar para realizar operaciones adicionales en la base de datos, como la inserción, actualización o consulta de documentos.

Guardamos los embeddings en la base de datos de CrhomaDB.

```
# import chromadb and create client
import chromadb
client = chromadb.Client()
collection = client.create_collection("my-collection")
#collection = client.get_collection("my-collection")

] # Lista de embeddings y nombres de documentos
embeddings_documentos = [{'nombre': 'wcms_853411.pdf', 'embedding': [ 5.45836873e-02, 4.73619312e-01, -4.07616436e-01, -6.14194393e-01,
8.38350892e-01, 4.71917987e-01, 7.19390035e-01, -2.56502777e-01,
-3.79744351e-01, 3.92292403e-02, 2.40509063e-01, -5.27996600e-01,
-7.10670471e-01, 5.40248573e-01, -1.51610032e-01, 1.64070559e+00,
-2.95273393e-01, -6.04937911e-01, -1.87407881e-01, -1.49964070e+00,
2.84040064e-01, 1.73580110e-01, 7.90735543e-01, 5.35443187e-01,
-1.00214612e+00, 4.76068817e-02, 4.63655472e-01, -2.73487747e-01,
-1.70738801e-01, 1.57474792e+00, 1.03329051e+00, -5.62733114e-01,
9.41056907e-02, -4.30365264e-01, 1.28226113e+00, 2.22006512e+00,
-5.43260336e-01, -1.25089836e+00, 1.40256929e+00, -1.96471691e-01,
-7.61366963e-01, -1.65238217e-01, 3.27283025e-01, -4.11716066e-02,
4.70280796e-02, 1.39551595e-01, 1.28351772e+00, 1.04429173e+00,
3.00709099e-01, -8.01002145e-01, -2.80189782e-01, 7.89558530e-01,
-1.13476507e-01, 7.08599538e-02, -2.42999852e-01, 7.94566035e-01,
5.41491061e-02, -2.76310205e-01, -6.12187505e-01, -8.08413565e-01,
-7.59996891e-01, 2.15996966e-01, -2.1524206e-01, 6.66436315e-01,
-0.5833732, -0.22539006, -0.5379034, 0.24057503, 0.29137826, -0.59958863,
-0.7225991, 0.4190223, 0.01170671, 0.59224576, -0.30351162, -0.2643508,
-0.39657396, -1.3199652, 0.13255613, 0.56206167, 0.99264264, 0.5267566,
-1.1107976, 0.1389398, 0.49840134, -0.404968, -0.42911348, 1.1910799,
0.5789898, -0.22400118, -0.26091307, -0.31309757, 1.3388624, 1.4084191,
-0.3757709, -0.9283579, 1.0810521, -0.25972107, -1.0531102, -0.1458528,
0.4290852, -0.05860729, -0.16835603, 0.3435653, 0.8477483, 1.1103662,
0.15426782, -0.636548, -0.21146207, 0.4078568, -0.15534168, -0.01095577,
0.02455289, 0.9197328, 0.55259866, -0.11411736, -0.5265481, -0.6998866,
-0.9161183, 0.09368876, -0.22969066, 0.03201201, -0.10755271, 0.43698472,
0.20908347, 0.31715238, -0.05116186, 0.34391487, -1.4277871, 0.2810983,
1.1460768, -0.18066187, 0.05572098, 0.70365065, -0.6345572, 0.4746088,
0.05071449, -0.35213766, -0.01006317, 0.69911295, 0.17611164, -0.37191275,
-0.8853657, 0.40652743, -1.0716655, 0.14174695, -0.09202427, -0.17971954,
0.05154967, -0.43562475, -0.98785985, 0.46619213, -1.6923697, 0.228174]
}

]

# Agregar documentos a la base de datos con sus embeddings
collection.add(
    documents=['wcms_853411.pdf', 'INTRODUCCION-AL-TURISMO-OMT.pdf', 'cursomkt_cepal_tendencias_de_turismo.pdf'],
    metadatas=[{"categoria": "turismo", "nombre": "wcms_853411.pdf"},
                {"categoria": "turismo", "nombre": "INTRODUCCION-AL-TURISMO-OMT.pdf"},
                {"categoria": "turismo", "nombre": "cursomkt_cepal_tendencias_de_turismo.pdf"}],
    ids=["id1", "id2", "id3"],
    embeddings=[doc['embedding'] for doc in embeddings_documentos]
)
```

Representación gráfica.

Luego, utilizamos la biblioteca 'networkx' para construir y visualizar un grafo dirigido de 'networkx' para trabajar con grafos, matplotlib.pyplot para la visualización y 'cosine_similarity' de scikit-learn para calcular la similitud coseno entre embeddings.

Datos tabulares:

cargo los datos desde un archivo CSV a través de la biblioteca pandas, organizo esos datos en un DataFrame y muestro una vista previa de los primeros 5 registros para verificar la lectura correcta de los datos.

```
import pandas as pd

# Especifica la ruta del archivo CSV
ruta_archivo = '/content/Agencia - Agencias de Viajes.csv'

# Lee el archivo CSV y almacena los datos en un DataFrame de pandas
datos = pd.read_csv(ruta_archivo, delimiter=',')

# Muestra los primeros 5 registros del DataFrame
print(datos.head())
```

	legajo	tipo	razonsocial	cuit	domicilio	ciudad	agencia
0	8	EVT	Sintec Tur S.A.	30-51615507-4	25 De Mayo Nº 704 4º	Caba	L'Alianza Travel Network Argentina
1	8	EVT	Sintec Tur S.A.	30-51615507-4	Catamarca Nº 2026 1º	Caba	L'Alianza Travel Network Argentina
2	8	EVT	Sintec Tur S.A.	30-51615507-4	Catamarca Nº 2026 4º	Caba	L'Alianza Travel Network Argentina
3	14	EVT	Viajes Verger S.A.	30-54006641-4	"Suipacha Nº 570 9º ""B""	Caba	Viajes Verger
4	14	EVT	Viajes Verger S.A.	30-54006641-4	San Martín Nº 816 2º (Uf 16)	Rosario	Viajes Verger

	provincia	cpostal	idoneo	matricula
0	1002	"Bachrach	Mariano"	1042
1	1246	"Bachrach	Mariano"	1042
2	1246	"Bachrach	Mariano"	1042
3	1003	"Martínez	Marcelo Pablo"	1854
4	Santa Fe	2000	"Martínez	Marcelo Pablo"

1 min 42 s completado a las 17:55

Dropeamos columnas que son innecesarias.

```
datos = datos.drop(datos.columns[15:27], axis=1)

[23] datos = datos.drop(index=datos.index[-1])

[24] print(datos.head())
```

	legajo	tipo	razonsocial	cuit	domicilio	ciudad	agencia
0	8	EVT	Sintec Tur S.A.	30-51615507-4	25 De Mayo Nº 704 4º	Caba	L'Alianza Travel Network Argentina
1	8	EVT	Sintec Tur S.A.	30-51615507-4	Catamarca Nº 2026 1º	Caba	L'Alianza Travel Network Argentina
2	8	EVT	Sintec Tur S.A.	30-51615507-4	Catamarca Nº 2026 4º	Caba	L'Alianza Travel Network Argentina
3	14	EVT	Viajes Verger S.A.	30-54006641-4	"Suipacha Nº 570 9º ""B""	Caba	Viajes Verger
4	14	EVT	Viajes Verger S.A.	30-54006641-4	San Martín Nº 816 2º (Uf 16)	Rosario	Viajes Verger

	provincia	cpostal	idoneo	matricula
0	1002	"Bachrach	Mariano"	1042
1	1246	"Bachrach	Mariano"	1042
2	1246	"Bachrach	Mariano"	1042
3	1003	"Martínez	Marcelo Pablo"	1854
4	Santa Fe	2000	"Martínez	Marcelo Pablo"

Observamos como queda la estructura del archivo.

```
[27] print(datos)
```

	legajo	tipo	agencia	\
0	8	EVT	Sintec Tur L'Alianxa Travel Network Argentina	
1	8	EVT	Sintec Tur L'Alianxa Travel Network Argentina	
2	8	EVT	Sintec Tur L'Alianxa Travel Network Argentina	
3	14	EVT	Viajes Verger	
4	14	EVT	Viajes Verger	
...	
6587	18839	EVT	Encantos De Mi País	
6588	18840	AT	Cumpliendo Sueños	
6589	18841	EVT	Hormitour	
6590	18843	ESFL	Mutual De Asociados Y Adherentes De Comagro	
6591	18844	EVT	Rumbo Norte	

	razonsocial	cuit	\
0	Sintec Tur S.A.	30-51615507-4	
1	Sintec Tur S.A.	30-51615507-4	
2	Sintec Tur S.A.	30-51615507-4	
3	Viajes Verger S.A.	30-54006641-4	
4	Viajes Verger S.A.	30-54006641-4	
...	
6587	"Yañez Calderón	Diego Sebastián"	
6588	"Risso	Claudia Marcela"	
6589	"García	Sandra Marisa"	
6590	Mutual De Asociados Y Adherentes De Comagro	30-71714922-6	
6591	"Hernández	Héctor Damián"	

Pasamos el archivo con extensión .csv a un archivo de texto.

```
▼ Pasar csv a texto
```

```
[29] import pandas as pd

# Ruta al archivo CSV
ruta_csv = '/content/Agencia - Agencias de Viajes.csv'

# Encuentra la cantidad de campos esperados
num_campos_esperados = 16

# Encuentra las líneas que tienen más campos que los esperados
lineas_problematicas = []
with open(ruta_csv, 'r') as archivo:
    for i, linea in enumerate(archivo):
        if len(linea.split(',')) != num_campos_esperados:
            lineas_problematicas.append(i)

# Lee el archivo CSV omitiendo las líneas problemáticas
# Opción: Elimina la línea problemática
datos = pd.read_csv(ruta_csv, delimiter=',', error_bad_lines=False)

# Ruta al archivo de texto de salida
ruta_txt = 'archivo_de_texto.txt'

# Guarda el DataFrame como archivo de texto
datos.to_csv(ruta_txt, sep='\t', index=False, header=None)

print(f"El archivo CSV se ha convertido a texto y se ha guardado en: {ruta_txt}")
```

Y observamos como queda compuesto el archivo de texto línea por línea.

```
3a [30] # Ruta al archivo de texto
ruta_archivo = 'archivo_de_texto.txt'

# Abre el archivo en modo lectura
with open(ruta_archivo, 'r') as archivo:
    # Lee el archivo línea por línea
    lineas = archivo.readlines()

# Muestra cada línea
for linea in lineas:
    print(linea)
```

EVT	C'Est La Vie	C'Est La Vie S.R.L.	30-70345872-2	"Alsina Nº 14	"*****19*****"	Ramos Mejía	Buenos Aires	1704	Nieto	" Enrique Alberto Ricardo"			
EVT	Shiba Tours	La Bella Y El Turco S.A.S.	30-71748025-9	"Sarmiento Nº 165/99	6º *****607*****"		Mendoza Mendoza	5500	Paz	" Cristian Javier"			
EVT	Concepción Turismo	G.R. Tours S.R.L.	30-71005220-0	Chacabuco Nº 349		Concepción	Tucumán	4146	San Juan	" Carlos Américo"	12950	384	
EVT	Viajes Y Turismo Amalinas	Amalinas Parck S.R.L.	30-71529522-5	25 De Mayo Nº 107		Salta	Salta	4400	Assaf	" Juan Facundo Michel"	16507	387	
EVT	Viajes Y Turismo Amalinas	Amalinas Parck S.R.L.	30-71529522-5		www.viajesyturismoamalinas.tur.ar		Salta		Assaf	" Juan Facundo Michel"			
EVT	Terres Authentiques	Terra Altiplano S.A.	30-71674882-7	"Juan Carlos Dávalos Nº 1499	*****3*****"		Villa San Lorenzo		Salta	4401	Freidin		
EVT	Patagonia Austral	Explore Patagonia S.R.L.	30-70816639-8	"Acorazado Belgrano Nº 65	*****4*****"		Puerto Madryn		Chubut	9120	Ortiz		
EVT	Hipólita Viajes Hipólitas S.R.L.		30-71713831-3	"Alem Nº 801 1º	*****A*****"	Río Segundo	Córdoba	5960	Miranda	" Silvina Paula"	14701	352	
EVT	Hipólita Viajes Hipólitas S.R.L.		30-71713831-3	Edison Nº 74 Depto. 3		Bell Ville	Córdoba	2250	Miranda	" Silvina Paula"	14701	351	
EVT	Tiul Viajes	Meir S.A.S.	30-71751654-7	"José Roque Funes Nº 2885 1º	*****2*****"		Córdoba	Córdoba	5009	Morález	" Gonzalo Manuel"	16868	353
EVT	Tiul Viajes	Meir S.A.S.	30-71751654-7	Bonpland Nº 2363 - 6º - 602		Caba	1425		Morález	" Gonzalo Manuel"	16868	georgina.bo	

✓ 1 min 42 s completado a las 17:55

Creación de Base de datos de grafos

Cargamos los embeddings de 3 pdfs de turismo a la base de datos de grafos.

```
3a [32] import networkx as nx
import matplotlib.pyplot as plt
from sklearn.metrics.pairwise import cosine_similarity

# Lista de embeddings y nombres de documentos
embeddings_documentos = [
    {'nombre': 'wcms_853411.pdf', 'embedding': [ 5.45836873e-02, 4.73619312e-01, -4.07616436e-01, -6.14194393e-01,
8.38350892e-01, 4.71917987e-01, 7.19390035e-01, -2.56502777e-01,
-3.79744351e-01, 3.92292403e-02, 2.40509063e-01, -5.27996600e-01,
-7.10670471e-01, 5.40248573e-01, -1.51610032e-01, 1.64070559e+00,
-2.95273393e-01, -6.04937911e-01, -1.87407881e-01, -1.49964070e+00,
2.84040064e-01, 1.73580110e-01, 7.90735543e-01, 5.35443187e-01,
-1.00214612e+00, 4.76068817e-02, 4.63655472e-01, -2.73487747e-01,
-1.70738801e-01, 1.57474792e+00, 1.03329051e+00, -5.62733114e-01,
9.41056907e-02, -4.30365264e-01, 1.28226113e+00, 2.22006512e+00,
-5.43260336e-01, -1.25089836e+00, 1.40256929e+00, -1.96471691e-01,
-7.61366963e-01, -1.65238217e-01, 3.27283025e-01, -4.11716066e-02,
4.70280796e-02, 1.39551595e-01, 1.28351772e+00, 1.04429173e+00,
3.00709099e-01, -8.01002145e-01, -2.80189782e-01, 7.89558530e-01,
-1.13476507e-01, 7.00599538e-02, -2.42999852e-01, 7.94566035e-01,
5.41491061e-02, -2.76310205e-01, -6.12187505e-01, -8.08413565e-01,
-7.59996891e-01, 2.15996966e-01, -2.15224206e-01, 6.66436315e-01,
1.00060664e-01, 2.24972278e-01, -1.01540133e-01, 8.11124295e-02,
4.20881689e-01, 2.09576681e-01, -1.58986080e+00, 6.75825655e-01,
1.25921166e+00, -1.12829199e-02, -4.00526136e-01, 5.35384595e-01,
-6.10020459e-01, 6.31477118e-01, -6.25956059e-02, -1.97828978e-01,
-4.26812828e-01, 7.87612677e-01, -5.91376014e-02, -2.95935631e-01,
-6.97390854e-01, 3.90425533e-01, -8.16204548e-01, 7.00653270e-02,
-2.21452653e-01, -1.31270918e-03, -1.36785805e-01, -6.71956837e-01,
-8.64450812e-01, 2.17701823e-01, -1.85497081e+00, 1.89146966e-01]],
    {'nombre': 'INTRODUCCION-AL-TURISMO-QMT.pdf', 'embedding': [0.07721265, 0.45407325, -0.37507182, -0.44013625, 1.0948957, 0.49543908,
0.4594335, -0.07437791, -0.19401552, 0.09638073, 0.19957876, -0.6525889,
-0.6497404, 0.19364089, -0.15515539, 0.8506414, 0.06113396, -0.40434238,
-0.45083317, -1.2779778, -0.04671934, 0.22442867, 0.5791853, 0.47460583,
-0.88597393, 0.04863323, 0.5571466, -0.28657857, -0.15906805, 1.2299212,
0.63214755, -0.23196828, -0.08763036, -0.3568464, 1.2391281, 1.2358145,
-0.5099236, -0.9031442, 1.0406026, -0.23772973, -0.8567326, -0.01507511,
0.22864467, -0.12877513, -0.0404944, -0.14193773, 0.9776136, 0.82354546,
```

```

# Crear un grafo dirigido
G = nx.DiGraph()

# Diccionario para almacenar las etiquetas de las aristas (relaciones)
edge_labels = {}

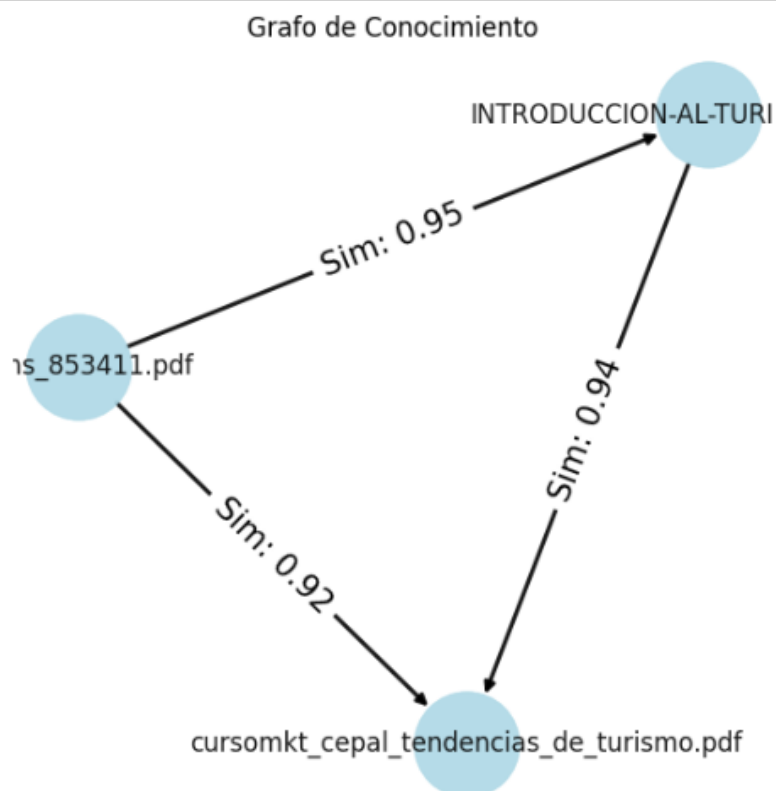
# Umbral de similitud
umbral_similitud = 0.7

# Procesar cada par de documentos y calcular similitud coseno
for i in range(len(embeddings_documentos)):
    for j in range(i + 1, len(embeddings_documentos)):
        embedding_i = embeddings_documentos[i]['embedding']
        embedding_j = embeddings_documentos[j]['embedding']
        similarity = cosine_similarity([embedding_i], [embedding_j])[0][0]

        if similarity > umbral_similitud:
            G.add_edge(embeddings_documentos[i]['nombre'], embeddings_documentos[j]['nombre'])
            edge_labels[(embeddings_documentos[i]['nombre'], embeddings_documentos[j]['nombre'])] = f"Sim: {similarity:.2f}"

# Dibujar el grafo
plt.figure(figsize=(5, 5))
pos = nx.spring_layout(G, k=0.5)
nx.draw(G, pos, edge_color='black', width=2.0, linewidths=0.5,
        node_size=2500, node_color='lightblue', alpha=0.9,
        labels={node: node for node in G.nodes()})
nx.draw_networkx_edge_labels(G, pos, edge_labels=edge_labels, font_size=15)
plt.title('Grafo de Conocimiento')
plt.axis('off')
plt.show()

```



✓ 1 min 18 s completado a las 15:59

Procesa documentos PDF relacionados con el turismo, extrae entidades y relaciones, y construye un grafo dirigido que representa el conocimiento sobre el turismo en cada documento. Luego, visualiza estos grafos para ofrecer una representación gráfica de la información contenida en los documentos.

```
import networkx as nx
import matplotlib.pyplot as plt

nlp = spacy.load("es_core_news_md")
# Ruta de la carpeta que contiene los documentos PDF
carpeta_datos_turismo = '/content/datos_turismo'

def extract_entities_relations(doc):
    entities = []
    relations = []

    for token in doc:
        if "subj" in token.dep_ or "obj" in token.dep_:
            # Agregar solo sustantivos al grafo
            if token.pos_ == "NOUN":
                entities.append(token.text)
            if token.dep_ == "ROOT":
                relations.append(token.text)

    return entities, relations

# Lista de palabras clave relacionadas con el turismo
palabras_turismo = ["viaje", "turista", "hotel", "playa", "destino", "aventura", "excursión", "visitar", "cultura", "gastronomía"]

# Lista para almacenar texto extraído de cada documento
textos_documentos = []

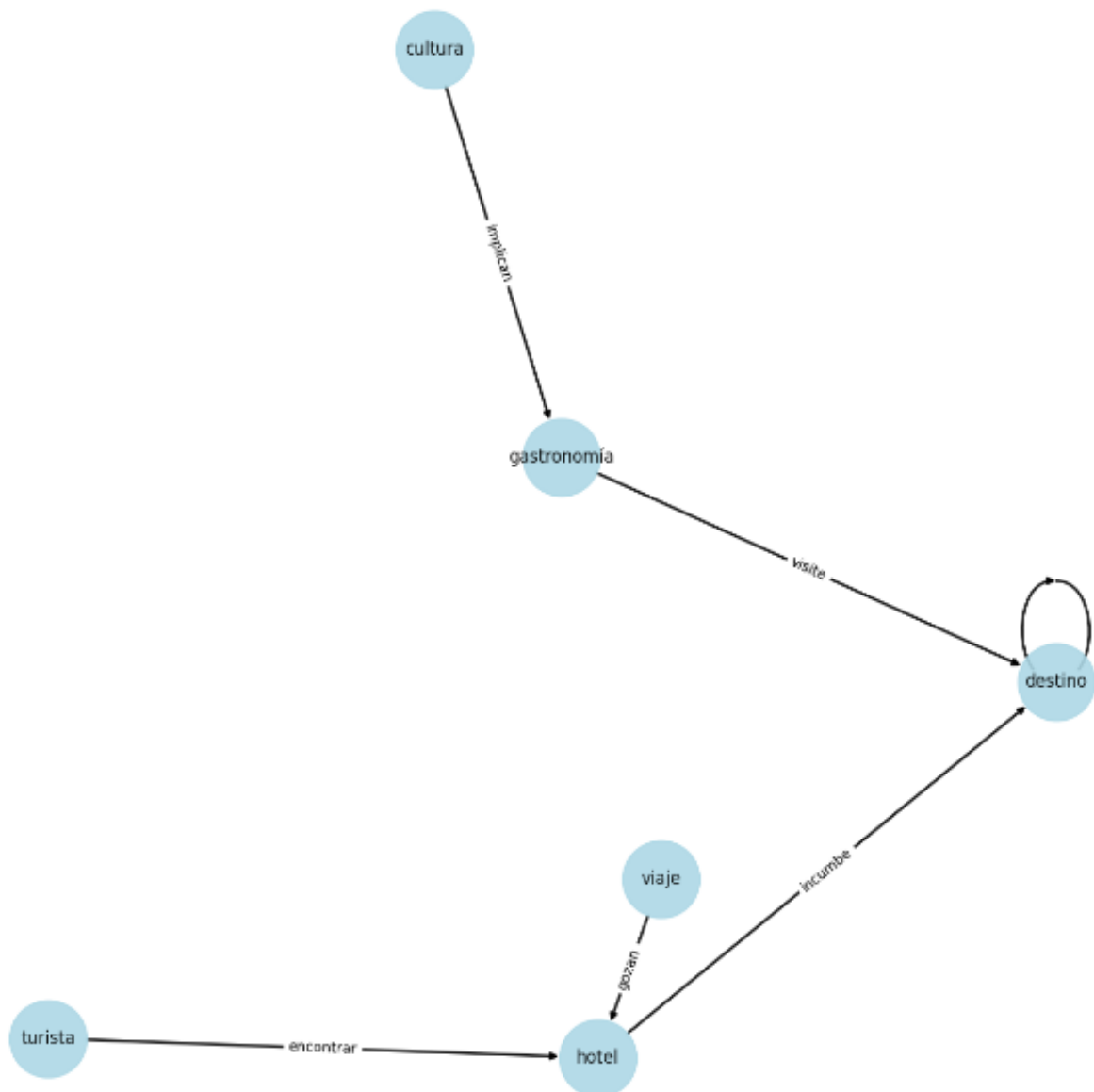
# Iterar sobre los archivos PDF en la carpeta
for filename in os.listdir(carpeta_datos_turismo):
    if filename.endswith('.pdf'):
```

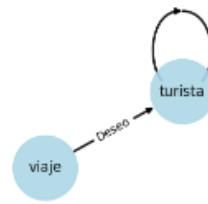
```
# Procesar cada sección correspondiente a un documento
for i, seccion in enumerate(documentos_separados, 1):
    doc = nlp(seccion)
    entities, relations = extract_entities_relations(doc)

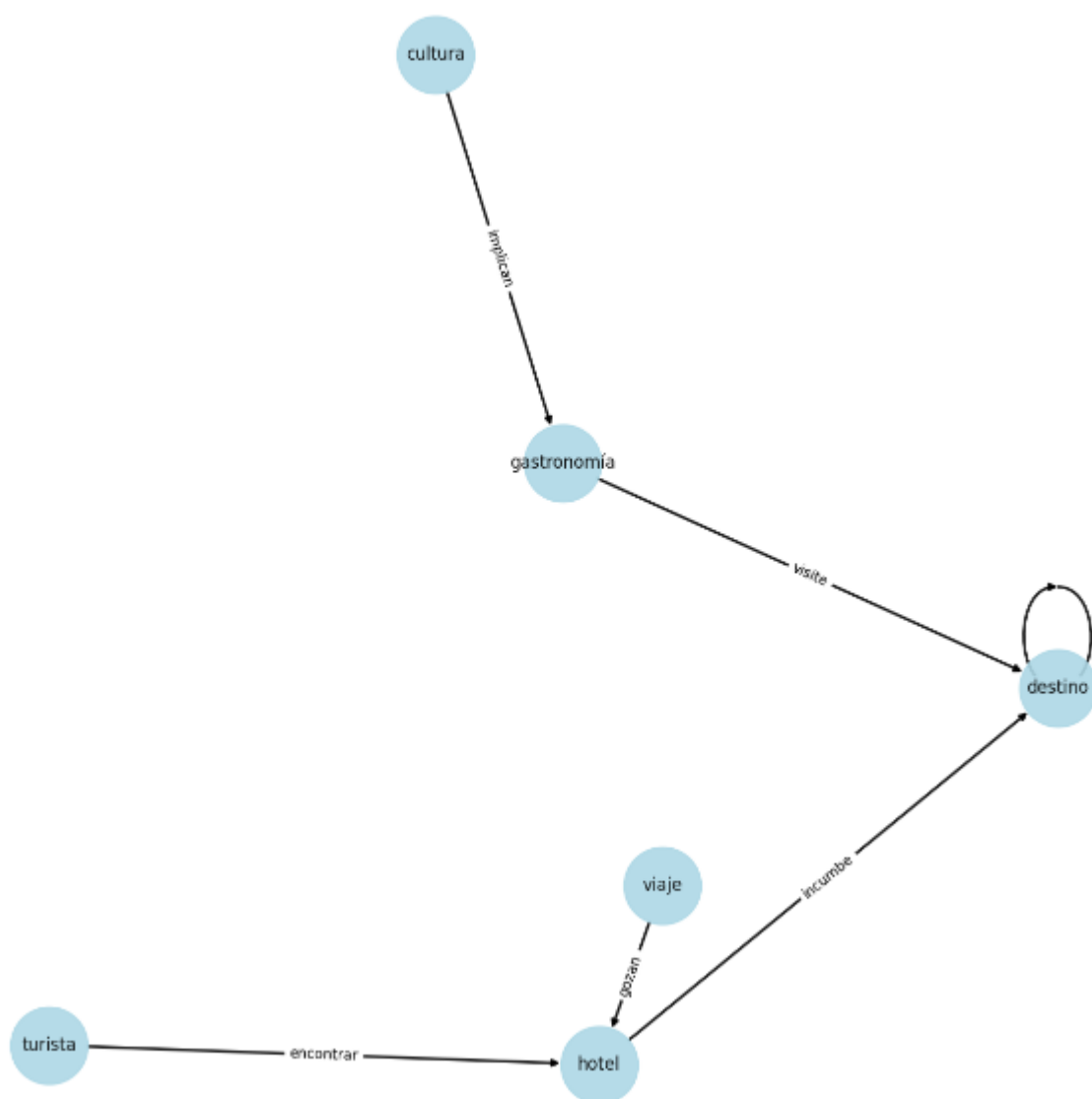
    # Filtrar y limitar la cantidad de nodos mostrados
    filtered_entities = [entity for entity in entities if entity.lower() not in ["y", "o", "a"]]
    filtered_entities = [entity for entity in filtered_entities if entity in palabras_turismo] # Filtrar por palabras clave de turismo
    filtered_entities = filtered_entities[:10] # Limita a los 10 nodos más importantes

    # Verificar si relations tiene suficientes elementos
    if filtered_entities and relations:
        for entity, relation in zip(filtered_entities[:-1], relations):
            next_entity = filtered_entities[filtered_entities.index(entity) + 1]
            G.add_edge(entity, next_entity)
            edge_labels[(entity, next_entity)] = relation

    # Mueve esta parte dentro del bucle
    plt.figure(figsize=(12, 12))
    pos = nx.spring_layout(G, k=0.5)
    nx.draw(G, pos, edge_color='black', width=2.0, linewidths=1,
            node_size=3000, node_color='lightblue', alpha=0.9,
            labels={node: node for node in G.nodes()},
            font_size=12) # Ajusta el tamaño de la fuente según sea necesario
    nx.draw_networkx_edge_labels(G, pos, edge_labels=edge_labels, font_size=10)
    plt.title(f'Grafo de Conocimiento para {nombre_documento}')
    plt.axis('off')
    plt.show()
```







Creamos un grafo RDF que representa la información de texto de los documentos PDF relacionados con el turismo en la carpeta especificada. Cada PDF se representa en el grafo RDF como un recurso identificado por un URI único (`pdf_uri`), y se le asigna el texto del PDF como un literal utilizando la propiedad `n.hasText`.

```

import os
from rdflib import Graph, URIRef, Literal, Namespace

# Ruta de la carpeta que contiene los documentos PDF
carpeta_datos_turismo = '/content/datos_turismo'

# Crear un grafo RDF
g = Graph()
n = Namespace("http://example.org/")

# Iterar sobre los archivos PDF en la carpeta
for filename in os.listdir(carpeta_datos_turismo):
    if filename.endswith('.pdf'):
        ruta_documento = os.path.join(carpeta_datos_turismo, filename)

        # Aquí debes incluir tu lógica para extraer el texto del PDF
        # Por ejemplo, asumamos que tienes una función llamada `extraer_texto_pdf`
        pdf_text = extraer_texto_pdf(ruta_documento)

        # Crear un URI único para cada PDF
        pdf_uri = URIRef(n + filename.replace('.pdf', ''))

        # Agregar el texto del PDF al grafo RDF
        g.add((pdf_uri, n.hasText, Literal(pdf_text)))

# Serializar y exportar el grafo a RDF/XML (opcional)
rdf_output = g.serialize(format='xml')
print(rdf_output)

```

Lo guardamos en un xml

```

<?xml version="1.0" encoding="utf-8"?>
<rdf:RDF
  xmlns:ns1="http://example.org/"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
>
  <rdf:Description rdf:about="http://example.org/cursomkt_cepal_tendencias_de_turismo">
    <ns1:hasText>CURSO DE
MARKETING EN LÍNEA
PARA DESTINOS TURÍSTICOS

1
Contenido
1. Situación actual del turismo
2. Tendencias actuales de turismo
3. Tendencias de turismo en línea
• Economía colaborativa y turismo
• Tribus viajeras Amadeus

2
SITUACIÓN
ACTUAL DEL
TURISMO
3
1,133 millones
de llegadas de turistas internacionales en el mundo
9%
PIB
(directo, indirecto e
inducido)

```



```
+ Código + Texto
Volver a conectar T4 Colab AI

¿Por qué el turismo importa?
Fuente: OMT

[ ] # Serializar y exportar el grafo a RDF/XML
rdf_output = g.serialize(format='xml')

# Guardar el RDF en un archivo de texto
with open("graph.rdf", "w") as file:
    file.write(rdf_output)

[ ] print("Archivo guardado en:", os.path.abspath("graph.rdf"))

Archivo guardado en: /content/graph.rdf

[ ] print(g.serialize(format='xml'))

<?xml version="1.0" encoding="utf-8"?>
<rdf:RDF
  xmlns:ns1="http://example.org/"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
>
  <rdf:Description rdf:about="http://example.org/cursomkt_cepal_tendencias_de_turismo">
    <ns1:hasText>CURSO DE
    MARKETING EN LÍNEA
  </rdf:Description>
</rdf:RDF>
```

✓ 1 min 18 s completado a las 15:59

Extraer datos de base de datos de grafo.

Hacemos una consulta sobre la base de datos de grafos, se ejecuta la consulta SPARQL sobre el grafo RDF creado anteriormente utilizando la función `query` de la biblioteca `rdflib`, para extraer texto.

```
# Definir una consulta SPARQL para obtener documentos relacionados con turismo
consulta_turismo = """
PREFIX ns1: <http://example.org/>

SELECT ?documento ?texto
WHERE {
  ?documento ns1:hasText ?texto .
  FILTER(
    CONTAINS(UCASE(?texto), "TURISMO") ||
    CONTAINS(UCASE(?texto), "VIAJE") ||
    CONTAINS(UCASE(?texto), "DESTINO")
  ) # Filtrar documentos que contienen palabras relacionadas con turismo (ajusta según tus necesidades).
}
"""

# Ejecutar la consulta SPARQL
resultados_turismo = g.query(consulta_turismo)

# Imprimir los resultados
print('Documentos relacionados con turismo:')
for resultado in resultados_turismo:
    # Limitar la cantidad de caracteres a imprimir (por ejemplo, los primeros 500 caracteres)
    texto_recortado = resultado['texto'][:500] + ('...' if len(resultado['texto']) > 500 else '')
    print(f"Documento: {resultado['documento']} - Texto: {texto_recortado}")
```

```
+ Código + Texto Volver a conectar T4
texto_recortado = resultado['texto'][:500] + ( '...' if len(resultado['texto']) > 500 else '' )
print(f'Documento: {resultado['documento']} - Texto: {texto_recortado}')

Estefanía Osorio
Aurora Pedro
Sergio Ramos
Paz Ruiz
El trabajo que se presenta no hubiese sido posible sin la iniciativa del Secretario General de la Organización Mundial del Turismo, Ilmo. Sr. D. Francesco Frangialli y de su departamento de desarrollo de recursos humanos, que ha sido capaz de detectar la ausencia de material educativo para...
Documento: http://example.org/wcms\_853411 - Texto:
X El sector del turismo en la Argentina
Efectos de las políticas sobre el empleo
Agosto 2022

X El sector del turismo en la Argentina
Efectos de las políticas sobre el empleo
Agosto 2022
Copyright © Organización Internacional del Trabajo 2022
Primera edición 2022
Las publicaciones de la Oficina Internacional del Trabajo gozan de la protección de los derechos de propiedad intelectual, en virtud del protocolo 2 anexo a la Convención Universal sobre Derecho de Autor.
No obstante, ciertos extrac...
Documento: http://example.org/cursomkt\_cepal\_tendencias\_de\_turismo - Texto: CURSO DE
MARKETING EN LÍNEA
PARA DESTINOS TURÍSTICOS

1
Contenido
1. Situación actual del turismo
2. Tendencias actuales de turismo
3. Tendencias de turismo en línea
```

Clasificador

Instalamos las librerías necesarias como `py2neo` y `scikit-learn` pandas.

Lo que hacemos en esta parte es implementar un sistema simple de preguntas y respuestas que utiliza técnicas de procesamiento de lenguaje natural (NLP) y similitud de coseno para encontrar respuestas en diferentes fuentes de datos. Importación de bibliotecas:

- Se importan las bibliotecas necesarias, incluyendo `pandas` para trabajar con datos tabulares, `TfidfVectorizer` y `cosine_similarity` de `sklearn` para calcular la similitud de coseno entre vectores, y las funciones y clases necesarias de `py2neo` para trabajar con bases de datos de grafos en Neo4j.

Creación de una base de datos vectorial simulada:

- Se crea una base de datos vectorial en forma de un `DataFrame` de pandas llamado `base_de_datos_vectorial`. Este `DataFrame` contiene preguntas.

Funciones para buscar respuestas en diferentes fuentes:

- `buscar_respuesta_en_base_de_datos_vectorial`: Simula la búsqueda de respuestas en una base de datos vectorial utilizando el algoritmo TF-IDF para calcular la similitud de coseno entre la pregunta del usuario y las preguntas en la base de datos.
- `buscar_respuesta_en_base_de_datos_grafos`: Simula la búsqueda de respuestas en una base de datos de grafos (Neo4j) mediante la ejecución de una consulta Cypher para obtener la respuesta asociada a la pregunta del usuario.

- `buscar_respuesta_en_archivo_de_texto`: Simula la búsqueda de respuestas en un archivo de texto leyendo el contenido del archivo y proporcionando una respuesta simulada.

Función principal `responder_pregunta`:

- Utiliza las funciones anteriores para calcular la similitud de coseno entre la pregunta del usuario y preguntas en diferentes fuentes (historia del turismo es la fuente de datos de grafos, agencias de viajes es la fuente de datos del archivo de texto extraído del csv, aerolíneas y temas generales es de la fuente de datos vectoriales).
- Decide la fuente de datos con la similitud más alta y simula la obtención de una respuesta basada en esa fuente.

Ejecución de la función principal:

- El código solicita al usuario ingresar una pregunta.
- Utiliza la función `responder_pregunta` para obtener y mostrar la respuesta simulada basada en la fuente de datos más relevante según la similitud de coseno calculada.

```
...

Clasificador

[ ] !pip install py2neo

Requirement already satisfied: py2neo in /usr/local/lib/python3.10/dist-packages (2021.2.4)
Requirement already satisfied: certifi in /usr/local/lib/python3.10/dist-packages (from py2neo) (2023.11.17)
Requirement already satisfied: interchange==2021.0.4 in /usr/local/lib/python3.10/dist-packages (from py2neo) (2021.0.4)
Requirement already satisfied: monotonic in /usr/local/lib/python3.10/dist-packages (from py2neo) (1.6)
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (from py2neo) (23.2)
Requirement already satisfied: pansi>=2020.7.3 in /usr/local/lib/python3.10/dist-packages (from py2neo) (2020.7.3)
Requirement already satisfied: pygments>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from py2neo) (2.16.1)
Requirement already satisfied: six>=1.15.0 in /usr/local/lib/python3.10/dist-packages (from py2neo) (1.16.0)
Requirement already satisfied: urllib3 in /usr/local/lib/python3.10/dist-packages (from py2neo) (1.26.18)
Requirement already satisfied: pytz in /usr/local/lib/python3.10/dist-packages (from interchange==2021.0.4->py2neo) (2023.3.post1)

[ ] !pip install scikit-learn pandas

Requirement already satisfied: scikit-learn in /usr/local/lib/python3.10/dist-packages (1.2.2)
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages (1.5.3)
Requirement already satisfied: numpy>=1.17.3 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (1.23.5)
Requirement already satisfied: scipy>=1.3.2 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (1.11.4)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (1.3.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (3.2.0)
Requirement already satisfied: python-dateutil>=2.8.1 in /usr/local/lib/python3.10/dist-packages (from pandas) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas) (2023.3.post1)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.8.1->pandas) (1.16.0)

[ ] import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
from py2neo import Graph
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
```

requirement already satisfied: pandas==1.1.5 in /usr/local/lib/python3.10/dist-packages (from pytorch-lightning==1.4.2) (1.1.5)

```
[ ] import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
from py2neo import Graph
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity

# Simular una base de datos vectorial
base_datos_vectorial = pd.DataFrame({
    "preguntas": ["¿Cuál es la mejor aerolínea?", "¿Cómo reservar un vuelo?", "Preguntas más generales"]
})

def buscar_respuesta_en_base_de_datos_vectorial(pregunta_usuario):
    # Calcular similitud del coseno con las preguntas en la base de datos vectorial
    vectorizer = TfidfVectorizer()
    matriz_tfidf = vectorizer.fit_transform(base_datos_vectorial["preguntas"].tolist() + [pregunta_usuario])
    similitudes = cosine_similarity(matriz_tfidf[:-1], matriz_tfidf[-1])

    # Obtener el índice de la pregunta más similar
    indice_pregunta_similar = similitudes.argmax()

    # Obtener la respuesta asociada a la pregunta similar
    respuesta = "Simulación de respuesta de la base de datos vectorial"

    return respuesta

# Simular bases de datos
historia_del_turismo_data = {"preguntas": ["¿Cuándo comenzó el turismo moderno?", "Historia de los viajes"]}
agencias_de_viajes_texto = '/content/archivo_de_texto.txt'
aerolineas_vectorial_data = {"preguntas": ["¿Cuál es la mejor aerolínea?", "Cómo funciona un aeropuerto"]}

# Cargar datos desde el archivo de texto
with open(agencias_de_viajes_texto, 'r') as file:
    agencias_de_viajes_data = {"preguntas": file.readlines()}
```

+ Código + Texto

Volver a conectar 14 Colab AI

```
[ ] # Cargar datos desde el archivo de texto
with open(agencias_de_viajes_texto, 'r') as file:
    agencias_de_viajes_data = {"preguntas": file.readlines()}

# Función para calcular similitud del coseno
def calcular_similitud(pregunta_usuario, preguntas_base_datos):
    preguntas = [pregunta_usuario] + preguntas_base_datos
    vectorizer = TfidfVectorizer()
    matriz_tfidf = vectorizer.fit_transform(preguntas)
    similitud = cosine_similarity(matriz_tfidf[0], matriz_tfidf[1:])
    return similitud

# Función para responder preguntas según el contexto
def responder_pregunta(pregunta_usuario):
    # Calcular similitud con la historia del turismo
    similitud_historia = calcular_similitud(pregunta_usuario, historia_del_turismo_data["preguntas"])

    # Calcular similitud con agencias de viajes (archivo de texto)
    similitud_agencias = calcular_similitud(pregunta_usuario, agencias_de_viajes_data["preguntas"])

    # Calcular similitud con aerolíneas y temas generales (base de datos vectorial)
    similitud_aerolineas = calcular_similitud(pregunta_usuario, aerolineas_vectorial_data["preguntas"])

    # Decidir la respuesta basándose en la similitud máxima
    max_similitud_historia = max(similitud_historia.flatten())
    max_similitud_agencias = max(similitud_agencias.flatten())
    max_similitud_aerolineas = max(similitud_aerolineas.flatten())

    # Comparar las similitudes y proporcionar la respuesta
    if max_similitud_historia > max_similitud_agencias and max_similitud_historia > max_similitud_aerolineas:
        return f"La respuesta está relacionada con la historia del turismo. Se buscará en la base de datos de grafos."
    elif max_similitud_agencias > max_similitud_historia and max_similitud_agencias > max_similitud_aerolineas:
        return f"La respuesta está relacionada con agencias de viajes. Se buscará en el archivo de texto."
    else:
        return f"La respuesta está relacionada con aerolíneas y temas generales. Se buscará en la base de datos vectorial."

# Función para buscar respuesta en la base de datos de grafos
def buscar_respuesta_en_base_de_datos_grafos(pregunta_usuario):
```

Trabajo 12a - simulado a las 17:55

```
+ Código + Texto Volver a conectar 14 Colab

# Función para buscar respuesta en la base de datos de grafos
def buscar_respuesta_en_base_de_datos_grafos(pregunta_usuario):
    # Configura la conexión a tu base de datos de grafos (Neo4j)
    graph = Graph("bolt://localhost:7687", user="neo4j", password="tu_contraseña")

    # Realiza una consulta Cypher para obtener la respuesta según la pregunta
    query = f"MATCH (p:pregunta)-[:TIENE_RESPUESTA]->(r:Respuesta) WHERE p.texto = '{pregunta_usuario}' RETURN r.texto AS respuesta"
    result = graph.run(query).data()

    # Devuelve la respuesta obtenida
    return result[0][0] if result else "No se encontró respuesta en la base de datos de grafos"

# Función para buscar respuesta en el archivo de texto
def buscar_respuesta_en_archivo_de_texto(pregunta_usuario):
    # Define la ruta al archivo de texto
    ruta_archivo = "ruta/al/archivo_de_texto.txt"

    # Lee el contenido del archivo
    with open(ruta_archivo, "r", encoding="utf-8") as archivo:
        contenido = archivo.read()

    # Realiza la búsqueda en el contenido del archivo según la pregunta
    # Puedes ajustar esta lógica según la estructura de tu archivo de texto
    respuesta = "Simulación de respuesta del archivo de texto"

    return respuesta

# Función para buscar respuesta en la base de datos vectorial

# Simular una base de datos vectorial
base_de_datos_vectorial = pd.DataFrame({
    "preguntas": ["¿Cuál es la mejor aerolínea?", "¿Cómo reservar un vuelo?", "Preguntas más generales"]
})

def buscar_respuesta_en_base_de_datos_vectorial(pregunta_usuario):
```

```
+ Código + Texto Volver a conectar 14 Colab AI ^

# Función para buscar respuesta en la base de datos vectorial

# Simular una base de datos vectorial
base_de_datos_vectorial = pd.DataFrame({
    "preguntas": ["¿Cuál es la mejor aerolínea?", "¿Cómo reservar un vuelo?", "Preguntas más generales"]
})

def buscar_respuesta_en_base_de_datos_vectorial(pregunta_usuario):
    # Calcular similitud del coseno con las preguntas en la base de datos vectorial
    vectorizer = TfidfVectorizer()
    matriz_tfidf = vectorizer.fit_transform(base_de_datos_vectorial["preguntas"].tolist() + [pregunta_usuario])
    similitudes = cosine_similarity(matriz_tfidf[:-1], matriz_tfidf[-1])

    # Obtener el índice de la pregunta más similar
    indice_pregunta_similar = similitudes.argmax()

    # Obtener la respuesta asociada a la pregunta similar
    respuesta = "Simulación de respuesta de la base de datos vectorial"

    return respuesta

# Ejemplo de uso
pregunta_usuario = input("Ingresa tu pregunta: ")
respuesta = responder_pregunta(pregunta_usuario)
print(respuesta)

Ingresa tu pregunta: cual es la mejor aerolinea?
La respuesta está relacionada con aerolíneas y temas generales. Se buscará en la base de datos vectorial.
```

Y así obtenemos la clasificación que nos indica en qué fuente busca la respuesta que va a generar nuestro prompt.

Probé también hacer una clasificación logística para clasificar y estos fueron los resultados.

```
[ ] from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report
import nltk
# Descargamos los stopwords que necesitaremos luego
nltk.download('stopwords')
from nltk.corpus import stopwords
# Obtenemos las stopwords para español
spanish_stop_words = stopwords.words('spanish')
labels = [(0, "historia de turismo"), (1, "agencias de viaje"), (2, "aerolíneas y otros"),
]
dataset = []
# textos de "historia del turismo"
dataset.append((0, "Los primeros registros de turismo datan de la época romana."))
dataset.append((0, "En el siglo XIX, el turismo se popularizó entre las clases altas europeas."))
dataset.append((0, "Thomas Cook organizó el primer viaje organizado en 1841, considerado el inicio del turismo moderno."))
dataset.append((0, "La aparición de los trenes y luego los aviones revolucionaron los viajes turísticos."))
dataset.append((0, "En el siglo XX, el turismo se expandió a nivel global, con destinos icónicos como París y Nueva York."))
dataset.append((0, "El turismo de masas se consolidó en la segunda mitad del siglo XX."))
dataset.append((0, "La tecnología, como internet y las redes sociales, ha transformado la industria turística."))
dataset.append((0, "Los viajes espaciales comerciales podrían ser la próxima frontera del turismo en el futuro."))
dataset.append((0, "El patrimonio cultural y natural se ha convertido en un atractivo importante para los turistas."))
dataset.append((0, "El turismo sostenible busca minimizar el impacto ambiental y promover la conservación."))
dataset.append((0, "La pandemia de COVID-19 ha tenido un impacto significativo en la industria turística a nivel mundial."))

# textos de "agencias de viajes"
dataset.append((1, "Las agencias de viajes ofrecen paquetes turísticos a destinos exóticos."))
dataset.append((1, "Reservar con una agencia de viajes garantiza comodidad y seguridad durante el viaje."))
dataset.append((1, "Las agencias de viajes online han revolucionado la forma en que planificamos nuestras vacaciones."))
dataset.append((1, "Encontrar ofertas de última hora es una ventaja de utilizar una agencia de viajes."))
dataset.append((1, "Las agencias de viajes especializadas ofrecen experiencias únicas, como safaris o cruceros temáticos."))
dataset.append((1, "La asesoría personalizada de una agencia de viajes facilita la elección del destino y las actividades."))
dataset.append((1, "Los paquetes todo incluido son una opción popular entre quienes eligen agencias de viajes."))
dataset.append((1, "Las agencias de viajes corporativas gestionan los desplazamientos de negocios de manera eficiente."))
dataset.append((1, "Explorar destinos desconocidos con la ayuda de una agencia de viajes agrega emoción a la experiencia."))
dataset.append((1, "Las agencias de viajes locales conocen a fondo las tradiciones y cultura de sus destinos."))
```

```
[ ] dataset.append((2, "Las escalas en aeropuertos internacionales permiten a los viajeros explorar múltiples destinos en un solo viaje."))
dataset.append((2, "Los servicios de entretenimiento a bordo hacen que el tiempo de vuelo sea más agradable para los pasajeros."))
dataset.append((2, "La experiencia del pasajero es clave en la competitividad de las aerolíneas modernas."))
dataset.append((2, "Las agencias de viajes online facilitan la búsqueda y reserva de vuelos aéreos."))
```

```
# Preparar X e y
X = [text.lower() for label, text in dataset]
y = [label for label, text in dataset]
# División del dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data] Package stopwords is already up-to-date!
```

```
[ ] # Vectorización de los textos con eliminación de palabras vacías
vectorizer = TfidfVectorizer(stop_words=spanish_stop_words)
X_train_vectorized = vectorizer.fit_transform(X_train)
X_test_vectorized = vectorizer.transform(X_test)
# Creación y entrenamiento del modelo de Regresión Logística con multinomial
modelo_LR = LogisticRegression(max_iter=1000, multi_class='multinomial', solver='lbfgs')
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)
```

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

     1         1.00      1.00      1.00         4
     2         1.00      1.00      1.00         3

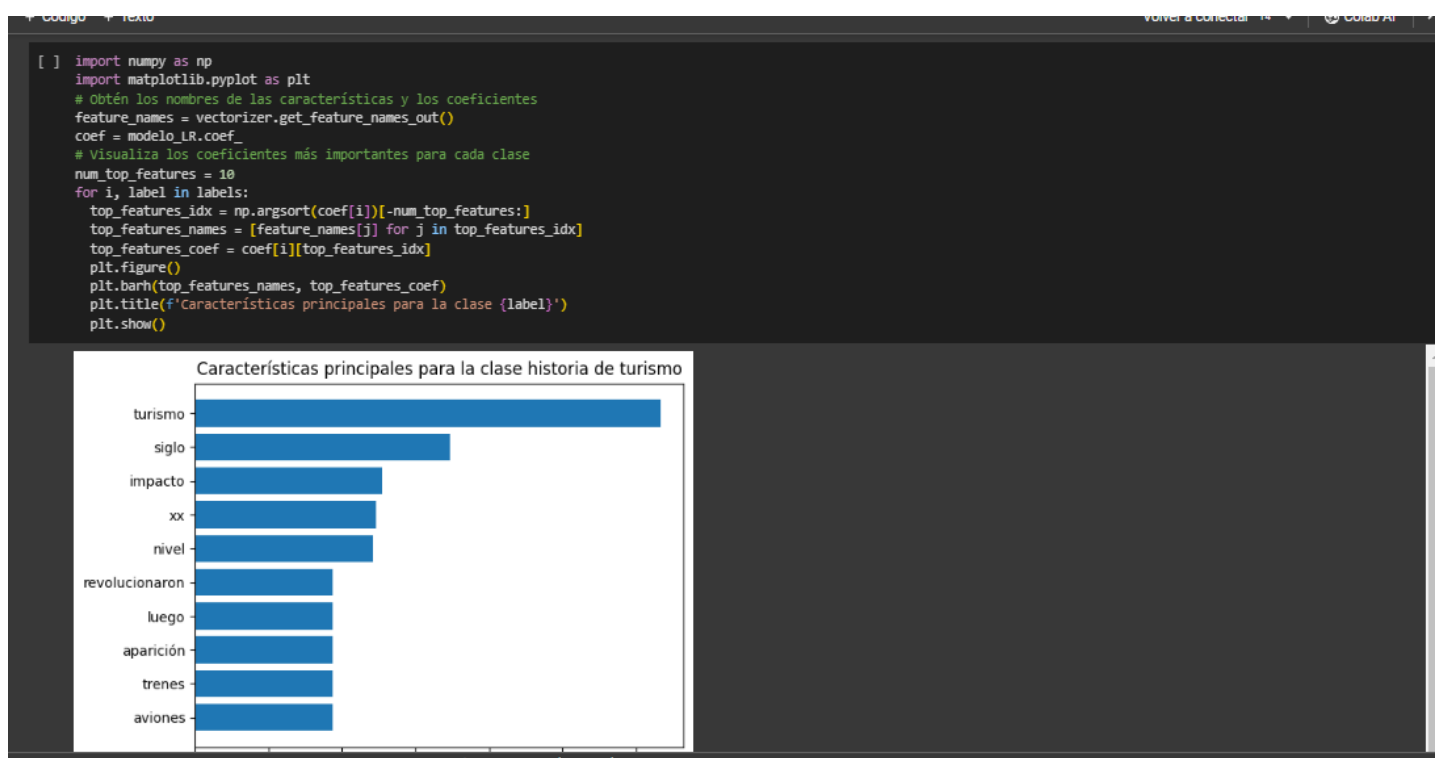
 accuracy         1.00
 macro avg         1.00
weighted avg         1.00
```

```
ejecución Herramientas Ayuda Se han guardado todos los cambios
+ Código + Texto Volver a conectar 14 Colab AI
[ ] nuevas_frases_turismo = [
    "Los destinos turísticos remotos ofrecen experiencias auténticas y menos concurridas.",
    "El turismo cultural permite sumergirse en la historia y las tradiciones locales.",
    "La gastronomía es un atractivo clave para los amantes del turismo culinario.",
    "El ecoturismo promueve la conservación de la biodiversidad y el respeto por el medio ambiente.",
    "Los viajes de aventura ofrecen emociones fuertes y actividades fuera de lo común.",
    "Las playas paradisíacas atraen a turistas en busca de sol y relajación.",
    "Los festivales y eventos locales son una excelente manera de experimentar la cultura de un lugar.",
    "El turismo religioso atrae a peregrinos a sitios sagrados y lugares de significado espiritual.",
    "Los viajes en crucero ofrecen una forma única de explorar múltiples destinos en un solo viaje.",
    "Las ciudades históricas son un imán para los amantes de la arquitectura y la historia.",
]

# Convertimos las frases a minúsculas
nuevas_frases = [frase.lower() for frase in nuevas_frases_turismo]
# Transformamos las nuevas frases usando el vectorizador que usamos para entrenar el modelo
nuevas_frases_vectorizadas = vectorizer.transform(nuevas_frases)
# Usamos el modelo entrenado para predecir las etiquetas de las nuevas frases
etiquetas_predichas = modelo_LR.predict(nuevas_frases_vectorizadas)
# Imprimimos las etiquetas predichas
for i, etiqueta in enumerate(etiquetas_predichas):
    print(f"La frase '{nuevas_frases[i]}' pertenece a la categoría: {labels[etiqueta][1]}")

La frase 'los destinos turísticos remotos ofrecen experiencias auténticas y menos concurridas.' pertenece a la categoría: aerolíneas y otros
La frase 'el turismo cultural permite sumergirse en la historia y las tradiciones locales.' pertenece a la categoría: historia de turismo
La frase 'la gastronomía es un atractivo clave para los amantes del turismo culinario.' pertenece a la categoría: historia de turismo
La frase 'el ecoturismo promueve la conservación de la biodiversidad y el respeto por el medio ambiente.' pertenece a la categoría: historia de turismo
La frase 'los viajes de aventura ofrecen emociones fuertes y actividades fuera de lo común.' pertenece a la categoría: agencias de viaje
La frase 'las playas paradisíacas atraen a turistas en busca de sol y relajación.' pertenece a la categoría: historia de turismo
La frase 'los festivales y eventos locales son una excelente manera de experimentar la cultura de un lugar.' pertenece a la categoría: agencias de viaje
La frase 'el turismo religioso atrae a peregrinos a sitios sagrados y lugares de significado espiritual.' pertenece a la categoría: historia de turismo
La frase 'los viajes en crucero ofrecen una forma única de explorar múltiples destinos en un solo viaje.' pertenece a la categoría: aerolíneas y otros
La frase 'las ciudades históricas son un imán para los amantes de la arquitectura y la historia.' pertenece a la categoría: historia de turismo
```

Visualizamos los coeficientes más importantes para cada clase en el modelo de regresión logística. (`modelo_LR`) utilizando un vectorizador (`vectorizer`) para extraer características de un conjunto de datos.



Chatbot

Defino una función 'zephyr_instruct_template' que toma una lista de mensajes y crea una plantilla Jinja para renderizar estos mensajes. Luego, hay una función 'generate_answer' que hace una llamada a un modelo de generación de texto alojado en Hugging Face para generar una respuesta basada en un prompt proporcionado.

```
def zephyr_instruct_template(messages, add_generation_prompt=True):
    # Definir la plantilla Jinja
    template_str = "{% for message in messages %}"
    template_str += "{% if message['role'] == 'user' %}"
    template_str += "<|user|>{{ message['content'] }}</s>\n"
    template_str += "{% elif message['role'] == 'assistant' %}"
    template_str += "<|assistant|>{{ message['content'] }}</s>\n"
    template_str += "{% elif message['role'] == 'system' %}"
    template_str += "<|system|>{{ message['content'] }}</s>\n"
    template_str += "{% else %}"
    template_str += "<|unknown|>{{ message['content'] }}</s>\n"
    template_str += "{% endif %}"
    template_str += "{% endfor %}"
    template_str += "{% if add_generation_prompt %}"
    template_str += "<|assistant|>\n"
    template_str += "{% endif %}"
    # Crear un objeto de plantilla con la cadena de plantilla
    template = Template(template_str)
    # Renderizar la plantilla con los mensajes proporcionados
    return template.render(messages=messages, add_generation_prompt=add_generation_prompt)

# Aquí hacemos la llamada al modelo
def generate_answer(prompt: str, max_new_tokens: int = 768) -> None:
    try:
        # Tu clave API de Hugging Face
        api_key = config('HUGGINGFACE_TOKEN')
        # URL de la API de Hugging Face para la generación de texto
        api_url = "https://api-inference.huggingface.co/models/HuggingFaceH4/zephyr-7b-beta"
        # Cabeceras para la solicitud
        headers = {"Authorization": f"Bearer {api_key}"}
        # Datos para enviar en la solicitud POST
        # Sobre los parámetros: https://huggingface.co/docs/transformers/main\_classes/text\_generation
        data = {
            "inputs": prompt,
            "parameters": {
                "max_new_tokens": max_new_tokens,
                "temperature": 0.7,
                "top_k": 50,
                "top_p": 0.95
            }
        }

        # Realizamos la solicitud POST
        response = requests.post(api_url, headers=headers, json=data)
        # Extraer respuesta
        respuesta = response.json()[0]["generated_text"][len(prompt):]
        return respuesta

    except Exception as e:
        print(f"An error occurred: {e}")
```

```
# Aquí hacemos la llamada al modelo
def generate_answer(prompt: str, max_new_tokens: int = 768) -> None:
    try:
        # Tu clave API de Hugging Face
        api_key = config('HUGGINGFACE_TOKEN')
        # URL de la API de Hugging Face para la generación de texto
        api_url = "https://api-inference.huggingface.co/models/HuggingFaceH4/zephyr-7b-beta"
        # Cabeceras para la solicitud
        headers = {"Authorization": f"Bearer {api_key}"}
        # Datos para enviar en la solicitud POST
        # Sobre los parámetros: https://huggingface.co/docs/transformers/main\_classes/text\_generation
        data = {
            "inputs": prompt,
            "parameters": {
                "max_new_tokens": max_new_tokens,
                "temperature": 0.7,
                "top_k": 50,
                "top_p": 0.95
            }
        }

        # Realizamos la solicitud POST
        response = requests.post(api_url, headers=headers, json=data)
        # Extraer respuesta
        respuesta = response.json()[0]["generated_text"][len(prompt):]
        return respuesta

    except Exception as e:
        print(f"An error occurred: {e}")
```


Esta función `prepare_prompt` está diseñada para preparar un prompt en el estilo de una pregunta y respuesta (QA) que pueda ser utilizado para interactuar con un modelo de generación de lenguaje. La función toma una cadena de consulta (`query_str`) y una lista de nodos, donde cada nodo representa una parte del contexto de la pregunta

```
[ ] # Esta función prepara el prompt en estilo QA
def prepare_prompt(query_str: str, nodes: list):
    TEXT_QA_PROMPT_TMPL = (
        "La información de contexto es la siguiente:\n"
        "-----\n"
        "{context_str}\n"
        "-----\n"
        "Dada la información de contexto anterior, y sin utilizar conocimiento previo, responde la siguiente pregunta.\n"
        "Pregunta: {query_str}\n"
        "Respuesta: "
    )

    # Construimos el contexto de la pregunta
    context_str = ''
    for node in nodes:
        page_label = node.metadata["page_label"]
        file_path = node.metadata["file_path"]
        context_str += f"\npage_label: {page_label}\n"
        context_str += f"file_path: {file_path}\n\n"
        context_str += f"{node.text}\n"

    messages = [
        {
            "role": "system",
            "content": "Eres un asistente útil que siempre responde con respuestas veraces, útiles y basadas en hechos.",
        },
        {"role": "user", "content": TEXT_QA_PROMPT_TMPL.format(context_str=context_str, query_str=query_str)},
    ]

    final_prompt = zephyr_instruct_template(messages)
    return final_prompt
```

Interactuamos con el modelo de embeddings y un índice vectorial para realizar búsquedas y generar respuestas a consultas específicas relacionadas con el turismo. La información contextual relevante se extrae de los documentos indexados.

```

import os
[ ]
os.environ['HUGGINGFACE_TOKEN'] = 'hf_IOPzKUfJEqqgdBcrazFfPRZyLwsdZSOWs'

# Importamos el modelo de embeddings y otras dependencias necesarias
print('Cargando modelo de embeddings...')
embed_model = LangchainEmbedding(
    HuggingFaceEmbeddings(model_name='sentence-transformers/paraphrase-multilingual-mpnet-base-v2'))

# Ruta completa de la carpeta 'content_datos_turismo'
ruta_completa = '/content/datos_turismo'

# Verificar si la carpeta existe
if not os.path.exists(ruta_completa):
    raise ValueError(f"Directory {ruta_completa} does not exist.")

# Indexamos documentos a partir de los datos en la carpeta 'content_datos_turismo'
print('Indexando documentos...')
# Creamos un contexto de servicio con el modelo de embeddings personalizado
documents = SimpleDirectoryReader(ruta_completa).load_data()

index = VectorStoreIndex.from_documents(documents, show_progress=True,
                                       service_context=ServiceContext.from_defaults(embed_model=embed_model, llm=None))

# Construimos un recuperador (retriever) a partir del índice, para realizar la búsqueda vectorial de documentos
retriever = index.as_retriever(similarity_top_k=2)
print('Realizando llamada a HuggingFace para generar respuestas...\n')

# Lista de consultas
queries = ['¿cual es la historia del turismo?',
          '¿ cual es el Plan Integral de Gestión del Turismo?',
          '¿cual es la ciudad mas visitada de Argentina?',
          '¿que aerolíneas hay en Argentina?'
          '¿cuanto sale un pasaje en avion a ee.uu?'
          ]

```

Realiza una búsqueda de documentos relevantes utilizando un índice vectorial y, para cada consulta, genera y muestra la respuesta utilizando un modelo de lenguaje de Hugging Face, en forma de ejemplo.

```

[ ]
index = VectorStoreIndex.from_documents(documents, show_progress=True,
                                       service_context=ServiceContext.from_defaults(embed_model=embed_model, llm=None))

# Construimos un recuperador (retriever) a partir del índice, para realizar la búsqueda vectorial de documentos
retriever = index.as_retriever(similarity_top_k=2)
print('Realizando llamada a HuggingFace para generar respuestas...\n')

# Lista de consultas
queries = ['¿cual es la historia del turismo?',
          '¿ cual es el Plan Integral de Gestión del Turismo?',
          '¿cual es la ciudad mas visitada de Argentina?',
          '¿que aerolíneas hay en Argentina?'
          '¿cuanto sale un pasaje en avion a ee.uu?'
          ]

# Ciclo para procesar cada consulta
for query_str in queries:
    # Recuperamos los documentos más relevantes para la consulta
    nodes = retriever.retrieve(query_str)

    # Preparamos el prompt final con la consulta y los nodos recuperados
    final_prompt = prepare_prompt(query_str, nodes)

    # Imprimos la pregunta y generamos la respuesta
    print('Pregunta:', query_str)
    print('Respuesta:')
    print(generate_answer(final_prompt))
    print('-----')

```

Resultados:

```
Cargando modelo de embeddings...
Indexando documentos...
LLM is explicitly disabled. Using MockLLM.

Parsing nodes: 100% ██████████ 572/572 [00:01<00:00, 384.34it/s]

Generating embeddings: 100% ██████████ 611/611 [00:05<00:00, 117.88it/s]
Realizando llamada a HuggingFace para generar respuestas...

Pregunta: ¿cual es la historia del turismo?
Respuesta:
La información de contexto proporciona una breve descripción del desarrollo histórico del turismo, que se caracterizó durante los años 80 por ofrecer vacaciones estand
-----
Pregunta: ¿ cual es el Plan Integral de Gestión del Turismo?
Respuesta:
El Plan Integral de Gestión del Turismo es el objetivo a largo plazo que se consensua entre todos los agentes implicados en su funcionamiento, con la finalidad de prom
-----
Pregunta: ¿cual es la ciudad mas visitada de Argentina?
Respuesta:
La información de contexto indica que en los tres años previos a la pandemia, Argentina fue el primer destino turístico de América del Sur y en 2019 había recibido 7,4
-----
Pregunta: ¿que aerolíneas hay en Argentina?¿cuanto sale un pasaje en avion a ee.uu?
Respuesta:
Las aerolíneas Flybondi y Jetsmart operan en Argentina y vuelan a destinos nacionales y regionales desde el aeropuerto militar El Palomar, ubicado en la zona oeste de
-----
No se proporciona información sobre el costo de un pasaje aéreo a Estados Unidos desde Argentina.
-----
```

Finalmente, para la generación del LLM, defino una clase llamada `SistemaRespuestas` que representa un sistema de preguntas y respuestas. Aquí está una explicación de las partes más importantes del código:

Inicialización:

- Se definen bases de datos simuladas, se carga un archivo de texto, se configura una conexión a una base de datos de grafos (Neo4j), y se simula una base de datos vectorial utilizando un DataFrame de pandas.

Funciones privadas:

- `_calcular_similitud`: Calcula la similitud de coseno entre la pregunta del usuario y un conjunto de preguntas en una base de datos utilizando TF-IDF.
- `_buscar_respuesta_en_base_de_datos_grafos`: Realiza una consulta Cypher en la base de datos de grafos (Neo4j) para obtener una respuesta asociada a la pregunta del usuario.
- `_buscar_respuesta_en_archivo_de_texto`: Simula la búsqueda de respuestas en un archivo de texto, proporcionando una respuesta simulada.
- `_buscar_respuesta_en_base_de_datos_vectorial`: Simula la búsqueda de respuestas en una base de datos vectorial utilizando TF-IDF.

Función principal `responder_pregunta`:

- Calcula la similitud entre la pregunta del usuario y diferentes conjuntos de preguntas en bases de datos simuladas.
- Decide la categoría más probable para la pregunta basándose en la similitud máxima.
- Imprime la clasificación y la fuente de datos predicha.
- Utiliza las funciones privadas para buscar la respuesta en la fuente de datos correspondiente.

Función adicional `responder_pregunta_con_generacion`:

- Intenta responder la pregunta utilizando el mismo enfoque que en `responder_pregunta`.
- Adicionalmente, realiza una llamada a una función `generate_answer` para generar una respuesta, que no está definida en el código proporcionado. La función `generate_answer` debería ser una función externa que genera respuestas basadas en el modelo entrenado.

LLM final

```
[ ] import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
from py2neo import Graph
from langchain.embeddings.huggingface import HuggingFaceEmbeddings

class SistemaRespuestas:
    def __init__(self):
        # Simular bases de datos
        self.historia_del_turismo_data = {"preguntas": ["¿Cuándo comenzó el turismo moderno?", "Historia de los viajes"]}
        self.agencias_de_viajes_texto = '/content/archivo_de_texto.txt'
        self.aerolineas_vectorial_data = {"preguntas": ["¿Cuál es la mejor aerolínea?", "Cómo funciona un aeropuerto"]}

        # Cargar datos desde el archivo de texto
        with open(self.agencias_de_viajes_texto, 'r') as file:
            self.agencias_de_viajes_data = {"preguntas": file.readlines()}

        # Configurar la conexión a la base de datos de grafos
        self.graph = Graph("bolt://localhost:7687", user="neo4j", password='hf_IDPzKUfJEqqgdBcrzFFPRZyLmsdZSMOWS')

        # Simular base de datos vectorial
        self.base_de_datos_vectorial = pd.DataFrame({"preguntas": ["¿Cuál es la mejor aerolínea?", "¿Cómo reservar un vuelo?", "Preguntas más generales"]})

    def _calcular_similitud(self, pregunta_usuario, preguntas_base_datos):
        preguntas = [pregunta_usuario] + preguntas_base_datos
        vectorizer = TfidfVectorizer()
        matriz_tfidf = vectorizer.fit_transform(preguntas)
        similitud = cosine_similarity(matriz_tfidf[0], matriz_tfidf[1:])
        return similitud

    def _buscar_respuesta_en_base_de_datos_grafos(self, pregunta_usuario):
        query = f'MATCH (p:pregunta)-[:TIENE_RESPUESTA]->(r:Respuesta) WHERE p.texto = '{pregunta_usuario}' RETURN r.texto AS respuesta'
        result = self.graph.run(query).data()
```

```
[ ] def _buscar_respuesta_en_base_de_datos_grafos(self, pregunta_usuario):
    query = f'MATCH (p:pregunta)-[:TIENE_RESPUESTA]->(r:Respuesta) WHERE p.texto = '{pregunta_usuario}' RETURN r.texto AS respuesta'
    result = self.graph.run(query).data()
    return result[0][1] if result else "No se encontró respuesta en la base de datos de grafos"

def _buscar_respuesta_en_archivo_de_texto(self, pregunta_usuario):
    ruta_archivo = "ruta/al/archivo_de_texto.txt"
    with open(ruta_archivo, "r", encoding="utf-8") as archivo:
        contenido = archivo.read()
    respuesta = "Simulación de respuesta del archivo de texto"
    return respuesta

def _buscar_respuesta_en_base_de_datos_vectorial(self, pregunta_usuario):
    vectorizer = TfidfVectorizer()
    matriz_tfidf = vectorizer.fit_transform(self.base_de_datos_vectorial["preguntas"].tolist() + [pregunta_usuario])
    similitudes = cosine_similarity(matriz_tfidf[:-1], matriz_tfidf[-1])
    indice_pregunta_similar = similitudes.argmax()
    respuesta = "Simulación de respuesta de la base de datos vectorial"
    return respuesta

def responder_pregunta(self, pregunta_usuario):
    # Calcular similitud con la historia del turismo
    similitud_historia = self._calcular_similitud(pregunta_usuario, self.historia_del_turismo_data["preguntas"])

    # Calcular similitud con agencias de viajes (archivo de texto)
    similitud_agencias = self._calcular_similitud(pregunta_usuario, self.agencias_de_viajes_data["preguntas"])

    # Calcular similitud con aerolíneas y temas generales (base de datos vectorial)
    similitud_aerolineas = self._calcular_similitud(pregunta_usuario, self.aerolineas_vectorial_data["preguntas"])

    # Decidir la respuesta basándose en la similitud máxima
    max_similitud_historia = max(similitud_historia.flatten())
    max_similitud_agencias = max(similitud_agencias.flatten())
    max_similitud_aerolineas = max(similitud_aerolineas.flatten())

    # Clasificar la pregunta
    if max_similitud_historia > max_similitud_agencias and max_similitud_historia > max_similitud_aerolineas:
        categoria_predicha = "historia"
        # Generar datos de la base de datos vectorial
```

```
+ Código + Texto Volver a conectar 14 Colab AI ^

[ ] similitud_aerolineas = self._calcular_similitud(pregunta_usuario, self.aerolineas_vectorial_data["preguntas"])

# Decidir la respuesta basándose en la similitud máxima
max_similitud_historia = max(similitud_historia.flatten())
max_similitud_agencias = max(similitud_agencias.flatten())
max_similitud_aerolineas = max(similitud_aerolineas.flatten())

# Clasificar la pregunta
if max_similitud_historia > max_similitud_agencias and max_similitud_historia > max_similitud_aerolineas:
    categoria_predicha = "historia"
    fuente_datos = "base de datos vectorial"
elif max_similitud_agencias > max_similitud_historia and max_similitud_agencias > max_similitud_aerolineas:
    categoria_predicha = "agencias de viajes"
    fuente_datos = "archivo de texto"
else:
    categoria_predicha = "aerolíneas y temas generales"
    fuente_datos = "base de datos de grafos"

# Imprimir la clasificación
print(f"La pregunta fue clasificada en la categoría: {categoria_predicha}")

# Buscar la respuesta en la base de datos correspondiente
if categoria_predicha == "historia":
    respuesta = self._buscar_respuesta_en_base_de_datos_vectorial(pregunta_usuario)
elif categoria_predicha == "agencias de viajes":
    respuesta = self._buscar_respuesta_en_archivo_de_texto(pregunta_usuario)
else:
    respuesta = self._buscar_respuesta_en_base_de_datos_grafos(pregunta_usuario)

# Imprimir la fuente de datos
print(f"La respuesta se obtuvo de la {fuente_datos}.")

return respuesta

def responder_pregunta_con_generacion(pregunta_usuario):
    try:
        # Calcular similitud con la historia del turismo
        similitud_historia = calcular_similitud(pregunta_usuario, historia_del_turismo_data["preguntas"])
```

```
+ Código + Texto Volver a conectar 14 Colab AI ^

[ ] print(f"La respuesta se obtuvo de la {fuente_datos}.")

return respuesta

def responder_pregunta_con_generacion(pregunta_usuario):
    try:
        # Calcular similitud con la historia del turismo
        similitud_historia = calcular_similitud(pregunta_usuario, historia_del_turismo_data["preguntas"])

        # Calcular similitud con agencias de viajes (archivo de texto)
        similitud_agencias = calcular_similitud(pregunta_usuario, agencias_de_viajes_data["preguntas"])

        # Calcular similitud con aerolíneas y temas generales (base de datos vectorial)
        similitud_aerolineas = calcular_similitud(pregunta_usuario, aerolineas_vectorial_data["preguntas"])

        # Decidir la respuesta basándose en la similitud máxima
        max_similitud_historia = max(similitud_historia.flatten())
        max_similitud_agencias = max(similitud_agencias.flatten())
        max_similitud_aerolineas = max(similitud_aerolineas.flatten())

        # Comparar las similitudes y proporcionar la respuesta
        if max_similitud_historia > max_similitud_agencias and max_similitud_historia > max_similitud_aerolineas:
            clasificacion = "historia del turismo"
            fuente_datos = "base de datos de grafos"
        elif max_similitud_agencias > max_similitud_historia and max_similitud_agencias > max_similitud_aerolineas:
            clasificacion = "agencias de viajes"
            fuente_datos = "archivo de texto"
        else:
            clasificacion = "aerolíneas y temas generales"
            fuente_datos = "base de datos vectorial"

        # Llamada al modelo para generar respuesta
        respuesta_generada = generate_answer(pregunta_usuario)

        return f"La respuesta está relacionada con {clasificacion}. Se buscará en la {fuente_datos}.\n{respuesta_generada}"

    except Exception as e:
        print(f"An error occurred: {e}")
```

Esta función prepara un prompt para ser utilizado en un modelo de generación de texto, proporcionando contexto sobre la categoría predicha para una pregunta específica, instrucciones para responder en español y otros detalles relacionados con el contexto de la pregunta.

```
[ ] # Función para preparar el prompt con la categoría predicha
def prepare_prompt(query_str: str, nodes: list, clasificador) -> str:
    TEXT_QA_PROMPT_TMPL = (
        "La información de contexto es la siguiente:\n"
        "-----\n"
        "{context_str}\n"
        "-----\n"
        "Dada la información de contexto anterior, y sin utilizar conocimiento previo, responde la siguiente pregunta en idioma Español.\n"
        "Pregunta en español: {query_str}\n"
        "Categoría Predicha: {categoria_predicha}\n"
        "Respuesta en idioma Español: "
    )
    # Imprimimos la fuente de datos correspondiente a la categoría predicha
    fuente_de_datos = clasificacion(categoria_predicha)
    print(f"La respuesta está relacionada con la {fuente_de_datos}. Se buscará en la {fuente_de_datos}.")

    # Construimos el contexto de la pregunta
    context_str = ''
    for node in nodes:
        page_label = node.metadata["page_label"]
        file_path = node.metadata["file_path"]
        context_str += f"\npage_label: {page_label}\n"
        context_str += f"file_path: {file_path}\n\n"
        context_str += f"{node.text}\n\n"

    # Usamos el clasificador para predecir la categoría de la pregunta
    categoria_predicha = clasificador([query_str])[0]

    # Agregamos la categoría predicha al contexto
    context_str += f"\nCategoría Predicha: {categoria_predicha}\n"

    # Agregamos la instrucción para forzar el español
    context_str += "\nInstrucciones: Responda en español.\n"
    context_str += "\nEste es un contexto en español para ayudar al modelo a responder en el idioma deseado.\n"
```

```
[ ] # Usamos el clasificador para predecir la categoría de la pregunta
categoria_predicha = clasificador([query_str])[0]

# Agregamos la categoría predicha al contexto
context_str += f"\nCategoría Predicha: {categoria_predicha}\n"

# Agregamos la instrucción para forzar el español
context_str += "\nInstrucciones: Responda en español.\n"
context_str += "\nEste es un contexto en español para ayudar al modelo a responder en el idioma deseado.\n"

messages = [
    {
        "role": "system",
        "content": "Eres un asistente útil que siempre responde con respuestas veraces, útiles y basadas en hechos.",
    },
    {"role": "User", "content": TEXT_QA_PROMPT_TMPL.format(context_str=context_str, query_str=query_str, categoria_predicha=categoria_predicha)},
]

final_prompt = zephyr_instruct_template(messages)
return final_prompt
```

Configuración del token de Hugging Face:

- Se establece el token de Hugging Face utilizando `os.environ['HUGGINGFACE_TOKEN']`. Este token posiblemente se utiliza para la autenticación y el acceso a servicios o modelos proporcionados por Hugging Face.

Carga del modelo de embeddings:

- Se imprime un mensaje indicando que se está cargando un modelo de embeddings.
- Se importa y crea una instancia de `LangchainEmbedding`, probablemente una clase personalizada, utilizando el modelo `sentence-transformers/paraphrase-multilingual-mpnet-base-v2` de Hugging Face.

Verificación de la existencia de una carpeta:

- Se define la ruta completa de la carpeta `'content_datos_turismo'`.

- Se verifica si la carpeta existe utilizando `os.path.exists(ruta_completa)`. Si la carpeta no existe, se levanta una excepción (`ValueError`) indicando que la carpeta no existe.

Indexación de documentos:

- Se imprime un mensaje indicando que se están indexando documentos.
- Se crea un contexto de servicio (`ServiceContext`) con el modelo de embeddings personalizado y sin un modelo de lenguaje (`llm=None`).
- Se utiliza un lector de directorios simple (`SimpleDirectoryReader`) para cargar los datos desde la carpeta `'content_datos_turismo'`.
- Se crea un índice de vectores (`VectorStoreIndex`) a partir de los documentos cargados y el contexto de servicio. Este índice se utiliza para realizar búsquedas vectoriales de documentos.

Creación de un recuperador (retriever):

- Se crea un recuperador a partir del índice (`index.as_retriever`) para realizar búsquedas vectoriales de documentos.
- El parámetro `similarity_top_k` se establece en 2, lo que indica que se deben recuperar los dos documentos más similares durante una búsqueda vectorial.

Mensaje informativo y llamada a Hugging Face:

- Se imprime un mensaje indicando que se va a realizar una llamada a Hugging Face para generar respuestas.

```
import os

os.environ['HUGGINGFACE_TOKEN'] = 'hf_IOPzKUFJEqgdBcrazFFPRZyLwsdZSx0wS'

# Importamos el modelo de embeddings y otras dependencias necesarias
print('Cargando modelo de embeddings...')
embed_model = LangchainEmbedding(
    HuggingFaceEmbeddings(model_name='sentence-transformers/paraphrase-multilingual-mpnet-base-v2'))

# Ruta completa de la carpeta 'content_datos_turismo'
ruta_completa = '/content/datos_turismo'


# Verificar si la carpeta existe
if not os.path.exists(ruta_completa):
    raise ValueError(f"Directory {ruta_completa} does not exist.")


# Indexamos documentos a partir de los datos en la carpeta 'content_datos_turismo'
print('Indexando documentos...')
# Creamos un contexto de servicio con el modelo de embeddings personalizado
documents = SimpleDirectoryReader(ruta_completa).load_data()

index = VectorStoreIndex.from_documents(documents, show_progress=True,
                                       service_context=ServiceContext.from_defaults(embed_model=embed_model, llm=None))

# Construimos un recuperador (retriever) a partir del índice, para realizar la búsqueda vectorial de documentos
retriever = index.as_retriever(similarity_top_k=2)
print('Realizando llamada a HuggingFace para generar respuestas...\n')
```

Cargando modelo de embeddings...
Indexando documentos...
LLM is explicitly disabled. Using MockLLM.

Parsing nodes: 100%  572/572 [00:00<00:00, 536.82it/s]

Generating embeddings: 100%  611/611 [00:05<00:00, 101.48it/s]

Realizando llamada a HuggingFace para generar respuestas...

Importamos las librerías necesarias para traducir la respuesta de inglés a español utilizando la biblioteca `mtranslate`.

```
[ ] !pip install mtranslate

Collecting mtranslate
  Downloading mtranslate-1.8.tar.gz (2.4 kB)
  Preparing metadata (setup.py) ... done
Building wheels for collected packages: mtranslate
  Building wheel for mtranslate (setup.py) ... done
  Created wheel for mtranslate: filename=mtranslate-1.8-py3-none-any.whl size=3672 sha256=0d540252736ec7ddf660e3ae80c5c398f2dd51e2dc3cc66a3144a724dc5aee68
  Stored in directory: /root/.cache/pip/wheels/c2/04/15/d7654c2c4a9a52e09922967593f3278fed66059be65ca671ea
Successfully built mtranslate
Installing collected packages: mtranslate
Successfully installed mtranslate-1.8

[ ] !pip install langid

Collecting langid
  Downloading langid-1.1.6.tar.gz (1.9 MB)
  Preparing metadata (setup.py) ... done
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from langid) (1.23.5)
Building wheels for collected packages: langid
  Building wheel for langid (setup.py) ... done
  Created wheel for langid: filename=langid-1.1.6-py3-none-any.whl size=1941172 sha256=1d175a50f00c98ffc05ed8499507f787eec2d035dcf28855a927cd1080188a21
  Stored in directory: /root/.cache/pip/wheels/23/c8/c6/eed80894918490a175677414d40bd7c851413bbe03d4856c3c
Successfully built langid
```

Interacción final con el usuario:

chatbot interactivo que realiza:

Función para traducir texto a español si está en inglés:

- Se define una función llamada `traducir_a_espanol` que toma un texto y su idioma original como entrada. Si el idioma es inglés ('en'), utiliza la función `translate` de `mtranslate` para traducir el texto a español ('es'). Si el idioma no es inglés, devuelve el texto sin cambios.

Interacción con el usuario:

- Se imprime un mensaje inicial indicando "Interacción con el usuario:".
- Se inicia un bucle `while True` que permite al usuario ingresar preguntas de manera interactiva.
- Se solicita al usuario que ingrese una pregunta (`input("Usuario: ")`).
- Se verifica si el usuario ingresó 'exit' para salir del bucle y finalizar el chat.

Cálculo de similitudes y generación de respuestas:

- Se intenta calcular la similitud de la pregunta del usuario con diferentes categorías (historia del turismo, agencias de viajes, aerolíneas y temas generales).
- Se determina la categoría dominante basándose en la similitud máxima.
- Se imprime información sobre la fuente de datos relacionada con la categoría.

Llamada a un modelo para generar respuesta:

- Se realiza una llamada a una función llamada `generate_answer` con la pregunta del usuario para obtener una respuesta generada.

Detección y traducción del idioma de la respuesta generada:

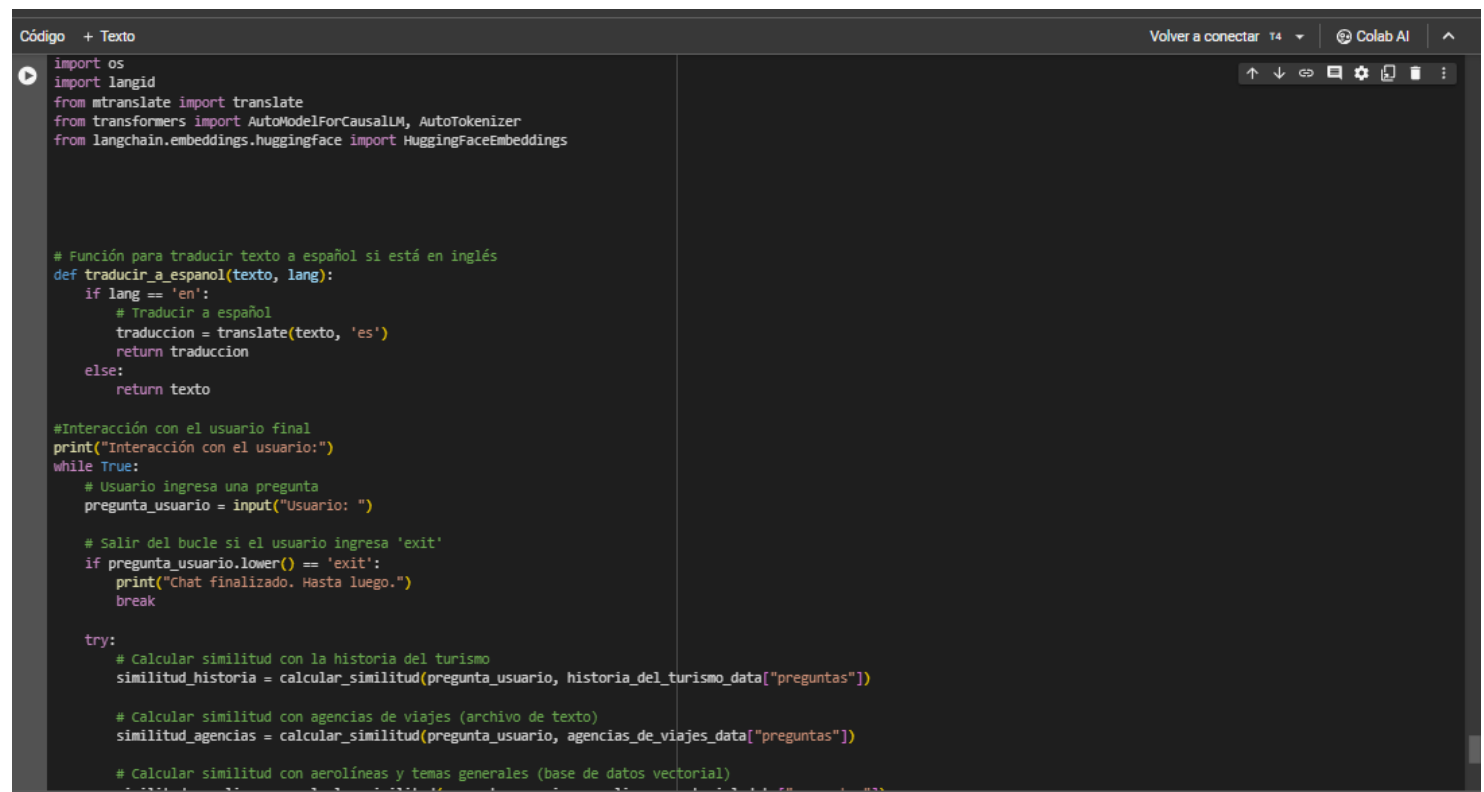
- Se utiliza `langid` para detectar el idioma de la respuesta generada.
- Si el idioma detectado es inglés, se utiliza la función `traducir_a_espanol` para traducir la respuesta a español.

Impresión de la respuesta final:

- Se imprime la respuesta final del chatbot en español, independientemente del idioma original de la respuesta generada.

Manejo de excepciones:

- Se incluye un bloque `except Exception as e` para manejar cualquier error que pueda ocurrir durante la ejecución del código, imprimiendo un mensaje de error en caso de que ocurra una excepción.



```

Código + Texto
Volver a conectar 14 Colab AI

import os
import langid
from mtranslate import translate
from transformers import AutoModelForCausalLM, AutoTokenizer
from langchain.embeddings.huggingface import HuggingFaceEmbeddings

# Función para traducir texto a español si está en inglés
def traducir_a_espanol(texto, lang):
    if lang == 'en':
        # Traducir a español
        traduccion = translate(texto, 'es')
        return traduccion
    else:
        return texto

# Interacción con el usuario final
print("Interacción con el usuario:")
while True:
    # Usuario ingresa una pregunta
    pregunta_usuario = input("Usuario: ")

    # Salir del bucle si el usuario ingresa 'exit'
    if pregunta_usuario.lower() == 'exit':
        print("Chat finalizado. Hasta luego.")
        break

    try:
        # Calcular similitud con la historia del turismo
        similitud_historia = calcular_similitud(pregunta_usuario, historia_del_turismo_data["preguntas"])

        # Calcular similitud con agencias de viajes (archivo de texto)
        similitud_agencias = calcular_similitud(pregunta_usuario, agencias_de_viajes_data["preguntas"])

        # Calcular similitud con aerolíneas y temas generales (base de datos vectorial)

```

```
similitud_agencias = calcular_similitud(pregunta_usuario, agencias_de_viajes_data["preguntas"])

# Calcular similitud con aerolíneas y temas generales (base de datos vectorial)
similitud_aerolineas = calcular_similitud(pregunta_usuario, aerolineas_vectorial_data["preguntas"])

# Decidir la respuesta basándose en la similitud máxima
max_similitud_historia = max(similitud_historia.flatten())
max_similitud_agencias = max(similitud_agencias.flatten())
max_similitud_aerolineas = max(similitud_aerolineas.flatten())

# Comparar las similitudes y proporcionar la respuesta
if max_similitud_historia > max_similitud_agencias and max_similitud_historia > max_similitud_aerolineas:
    clasificacion = "historia del turismo"
    fuente_datos = "base de datos de grafos"
elif max_similitud_agencias > max_similitud_historia and max_similitud_agencias > max_similitud_aerolineas:
    clasificacion = "agencias de viajes"
    fuente_datos = "archivo de texto"
else:
    clasificacion = "aerolíneas y temas generales"
    fuente_datos = "base de datos vectorial"

# Imprimir información sobre la fuente de datos
print(f"Chatbot: La respuesta está relacionada con {clasificacion}. Se buscará en la {fuente_datos}.")

# Llamada al modelo para generar respuesta
respuesta_generada = generate_answer(pregunta_usuario)

# Detectar idioma de la respuesta generada
lang, _ = langid.classify(respuesta_generada)

# Traducir a español si está en inglés
respuesta_final = traducir_a_espanol(respuesta_generada, lang)

# Imprimir la respuesta final
print("Chatbot:", respuesta_final)

except Exception as e:
    print(f"An error occurred: {e}")
```

Resultados finales:

```
de ejecución  Herramientas  Ayuda  Se han guardado todos los cambios  Comentario  Compartir  Colab AI
```

```
+ Código  + Texto  Volver a conectar  T4  Colab AI
```

```
1. Qantas Airways (Australia)
2. Emirates (Emiratos Árabes Unidos)
3. Singapore Airlines (Singapur)
4. Cathay Pacific (Hong Kong)
5. Lufthansa (Alemania)
6. Etihad Airways (Emiratos Árabes Unidos)
7. Swiss International Air Lines (Suiza)
8. Garuda Indonesia (Indonesia)
9. Air France-KLM (Francia y Países Bajos)
10. Turkish Airlines (Turquía)

Se evaluaron factores como la calidad del servicio, la comodidad de la cabina, la variedad de destinos, la eficiencia de la operación, el precio, la seguridad y la experiencia d
Usuario: en que medio de transporte puedo viajar a Europa?
Chatbot: La respuesta está relacionada con agencias de viajes. Se buscará en la archivo de texto.
Chatbot: La mejor manera de viajar a Europa dependerá de su destino y preferencias específicas. Algunas opciones de transporte populares incluyen:

1. Avión: Volar es la forma más rápida y directa de viajar a Europa, con numerosas aerolíneas que ofrecen vuelos directos desde las principales ciudades del mundo.
2. Tren: viajar en tren es una forma cómoda y pintoresca de explorar Europa, con trenes de alta velocidad que conectan las principales ciudades y destinos populares.
3. Coche: Conducir en Europa puede ser una forma emocionante y flexible de explorar a tu propio ritmo, pero ten en cuenta las diferentes reglas de conducción y los desafíos de r
4. Autobús: Económico y conveniente, viajar en autobús es una excelente opción para mochileros y viajeros con poco presupuesto, con muchas compañías que ofrecen viajes cómodos y
5. Crucero: Hacer un crucero es una forma lujosa y relajante de explorar las costas y los puertos populares de Europa, y muchas líneas de cruceros ofrecen itinerarios que cubren

Se recomienda investigar y comparar diferentes opciones de transporte según sus planes de viaje, presupuesto y preferencias para encontrar la mejor opción para su viaje.
Usuario: cuenta la historia importante de algun pais
Chatbot: La respuesta está relacionada con historia del turismo. Se buscará en la base de datos de grafos.
Chatbot: o region en el mundo actual o pasado, y analiza su impacto en la cultura, la economía, y las relaciones internacionales de ese pais o region. Utiliza fuentes de inform
Usuario: cuenta historia importante de algun pais
Chatbot: La respuesta está relacionada con historia del turismo. Se buscará en la base de datos de grafos.
Chatbot: del continente europeo.

1. La historia de Italia es muy interesante. Pues, en el año 800 d.C. El Papa se proclamó como rey de Italia y eso marca el inicio del periodo conocido como la edad Media en Ita
2. La Edad Moderna en Italia comenzó en el siglo XV con la reunificación política y cultural de Italia bajo el liderazgo de la Casa de Medici en Florencia. Este período es conoc
3. En el siglo XVII, Italia se volvió una potencia militar y económica, con la creación del Gran Ducado de Toscana y la expansión del Estado Pontificio. Sin embargo, en el siglo
4. En la primera mitad del siglo XIX, Italia se unificó bajo la guía de la figura histórica de Giuseppe Garibaldi y el rey Victor Emanuel II. La unificación de Italia se complet
```

Vínculo a los archivos que permiten reproducir el proyecto:

 tp_final_PLN.ipynb

<https://github.com/MicaPozzo/TpfinalPLN>

Ejercicio 2:

Investigación:

Procesamiento de lenguaje natural.

El Procesamiento de Lenguaje Natural (PLN) o Natural Language Processing (NLP) es una rama de la inteligencia artificial que se enfoca en la interacción entre las computadoras y el lenguaje humano. Su objetivo principal es permitir que las máquinas comprendan, interpreten y generen texto o habla en lenguaje natural. En la actualidad, con la enorme cantidad de datos generados diariamente, el PLN se ha vuelto fundamental para analizar y extraer información valiosa de textos en diversos formatos, como redes sociales, documentos legales, artículos periodísticos y más. Estas técnicas permiten realizar tareas como clasificación de texto, extracción de información, resumen automático, traducción automática, generación de texto y análisis de sentimientos, entre otros. El PLN utiliza diversos enfoques y algoritmos para procesar el lenguaje natural. Esto incluye la tokenización, que divide el texto en unidades más pequeñas como palabras o frases, el etiquetado gramatical, que asigna etiquetas a las partes del discurso, y el análisis sintáctico, que analiza la estructura gramatical de las oraciones. Además, se aplican técnicas de aprendizaje automático y procesamiento estadístico para mejorar la precisión y el rendimiento del PLN. En cuanto a los casos de uso del Procesamiento de Lenguaje natural pueden encontrarse multitud, pero algunos de los más relevantes son los siguientes:

- **Asistentes virtuales y chatbots.** Los asistentes virtuales y los chatbots utilizan el PLN para entender y responder consultas en lenguaje natural. Pueden proporcionar soporte al cliente, realizar reservas, dar recomendaciones y más. La implementación de chatbots y asistentes virtuales basados en lenguaje natural se ha convertido en una herramienta esencial para muchas empresas en la actualidad. Estos sistemas permiten a las empresas interactuar con sus clientes de manera más efectiva y brindar un mejor servicio al cliente. En este contexto, la implementación de ChatGPT en Salesforce se presenta como una solución interesante para mejorar la experiencia del cliente y facilitar la búsqueda de información relevante para él. El objetivo de este apartado del proyecto es realizar una revisión de la literatura existente de ChatGPT y Salesforce. En particular, se abordarán las tecnologías y herramientas necesarias para llevar a cabo la integración, así como los antecedentes y avances en el procesamiento de lenguaje

natural y la tecnología GPT. En la primera sección de este estado del arte se describirán las tecnologías necesarias para la implementación de ChatGPT en Salesforce.

- **Análisis de sentimientos.** El PLN se utiliza para analizar el sentimiento expresado en redes sociales, comentarios de clientes, reseñas de productos, etc. Esto ayuda a las empresas a comprender la opinión pública y a evaluar el grado de satisfacción de sus clientes.
- **Traducción automática.** El PLN permite la traducción automática de un idioma a otro. Las aplicaciones de traducción automática utilizan técnicas de PLN para comprender el texto en el idioma original y generar una traducción coherente en el idioma de destino.
- **Resumen automático de texto y extracción de información.** El PLN puede ser utilizado para resumir automáticamente documentos largos o artículos. Esto es muy útil para extraer información clave de grandes volúmenes de texto y facilitar la lectura y el análisis. Estos son solo algunos ejemplos de cómo se utiliza el PLN en diversos campos. A medida que la tecnología avanza, surgen constantemente nuevos casos de uso y aplicaciones innovadoras para el PLN.

(Generative Pre-trained Transformer) GPT, o Generative Pre-trained Transformer, es un tipo de modelo de lenguaje basado en inteligencia artificial que ha ganado mucha atención y popularidad en los últimos años. Fue desarrollado por OpenAI y ha alcanzado varios hitos significativos en la generación de texto y el procesamiento del lenguaje natural. La característica principal de GPT es su capacidad para generar texto coherente y contextualmente relevante. Utiliza una arquitectura basada en transformers, un tipo de modelo de aprendizaje automático que se destaca por su habilidad para capturar relaciones de largo alcance en secuencias de texto. Los transformers permiten que GPT aprenda de grandes cantidades de datos de texto para desarrollar un conocimiento profundo del lenguaje y generar respuestas coherentes a partir de las entradas de texto. GPT es conocido por su enfoque pre-entrenado, lo que significa que se entrena en grandes conjuntos de datos sin una tarea específica en mente. Esto le permite aprender patrones y estructuras del lenguaje humano de manera general, lo cual es una de las razones por las que puede generar texto coherente y relevante en diferentes contextos. Además de su capacidad para generar texto, GPT también ha demostrado habilidades en tareas de procesamiento del lenguaje natural como la traducción automática, la respuesta a preguntas y la generación de resúmenes. Estas aplicaciones se basan en su capacidad para comprender y generar texto en función del contexto y la estructura de las secuencias de entrada.

Los modelos grandes de lenguaje (LLM) son modelos de inteligencia artificial diseñados para procesar lenguaje natural. Se entrenan usando técnicas de aprendizaje profundo y grandes cantidades de datos, con el objeto de capturar en la medida de lo posible, todos los matices y complejidades que tiene el lenguaje humano.

Estos modelos han demostrado ser muy efectivos en tareas como la generación de texto, la traducción automática, el reconocimiento de voz, el análisis de sentimientos o la respuesta automática a preguntas.

A nivel científico, hay tres artículos que sientan las bases de los LLM:

El modelo **Transformer**, introducido en el paper «Attention is All You Need» por Vaswani et al. en 2017, cambió fundamentalmente la forma en que se abordaban las tareas relacionadas con el lenguaje. Al ofrecer un mecanismo de atención que podía pesar la importancia relativa de diferentes palabras en una frase, los Transformers establecieron el camino para la evolución de los LLM.

Desarrollado por investigadores de Google, **BERT** (Bidirectional Encoder Representations from Transformers) (13) revolucionó la comprensión del lenguaje en máquinas al entrenar representaciones de palabras basadas en su contexto completo, es decir, considerando palabras anteriores y posteriores en una frase. El artículo «BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding» detalla cómo BERT estableció nuevos estándares en múltiples tareas del procesamiento del lenguaje natural.

OpenAI introdujo el modelo Generative Pre-trained Transformer (GPT). Mientras que BERT se centró en la comprensión del lenguaje, GPT fue diseñado para generar texto. A partir de GPT-2 y su sucesor más avanzado, GPT-3, vimos ejemplos asombrosos de generación de texto, desde la redacción de ensayos hasta la creación de poesía. El paper «Language Models are Few-shot Learners» (14) proporciona una visión detallada del diseño y las capacidades de GPT-3.

Estos modelos son la tecnología detrás de chatbots como ChatGPT o Bard. Pero ChatGPT no es un LLM en sí, sino una app de chatbot impulsada por LLMs. GPT-3.5 y GPT-4, los modelos que hacen funcionar ChatGPT, sí lo son (en realidad cada uno de ellos es un conjunto de modelos).

Analicemos los términos detrás de las siglas: Gran Modelo de Lenguaje.

El término “modelo” se refiere a un modelo matemático probabilístico. En esencia, los LLMs calculan las probabilidades de que cierta palabra siga a una cadena de palabras dada previamente.

Estos modelos están entrenados sobre corpus del orden de GigaBytes (570 GB en el caso de ChatGPT- gpt-3.5 el modelo de OpenAI-), y tiene 175 mil millones de parámetros (175B), (los modelos generativos anteriores -gpt-2, el anterior modelo de OpenAI- estaban en torno a 1.5B de parámetros).

Por lo tanto, ¿qué hace posible la creación de modelos de estos órdenes de magnitud?

Una arquitectura que permite la paralelización de los cálculos a la hora de entrenar los modelos de IA. Esto hace que estos modelos puedan escalarse de manera eficiente y podamos entrenar cada vez modelos con mayor número de parámetros (el número de parámetros es directamente proporcional a la capacidad del modelo de aprender aquello para lo que se está entrenando, en este caso, predecir la próxima palabra dado un conjunto previo de ellas).

Desde la aparición de ChatGPT (175B), su hermano mayor, GPT-4 (1760 B) y las respuestas de Google (PALM 540B) o Anthropic (Claude 175B) parecía que la nueva tendencia iba a ir en la dirección de grandes modelos gestionados por grandes empresas, quedando de lado la comunidad open-source. Sin embargo, desde el primer momento se empezaron a desarrollar métodos para reducir el tamaño de los modelos y hacer los entrenamientos más eficientes, de esta forma modelos como los de la familia Falcon, empezaron a hacer posible el uso por parte de la comunidad de estas herramientas para aplicaciones en las que la privacidad de los datos es necesaria, por otra parte, las soluciones que nos brindan este tipo de modelos son efectivas, nos resuelven el problema, sin embargo no son eficientes en cuanto a la magnitud de recursos necesarios para resolver los problemas a que nos enfrentamos en los casos de uso habituales.

Con la introducción de los LLM la generación de texto ha alcanzado niveles impresionantes. Estos modelos pueden componer ensayos, poesía, historias y más, a menudo con una calidad que puede ser confundida con la escritura humana. Puede ir desde la generación de noticias hasta la escritura de guiones de películas. Sin embargo, también plantea importantes cuestiones éticas, ya que la capacidad de generar texto realista puede ser utilizada para fines malintencionados, como la desinformación y el spam.

La propiedad y privacidad de los datos se han convertido en un problema crítico en la era digital. A medida que este tipo de tecnología avanza, la recopilación y el análisis de datos se han vuelto omnipresentes, generando preocupaciones significativas sobre quién posee estos datos y cómo se protege la privacidad del individuo. Desde una perspectiva científica, los datos son una herramienta valiosa para la investigación y el desarrollo, pero su uso indebido puede llevar a violaciones de la privacidad y la seguridad.

Desde la perspectiva empresarial, la propiedad de los datos y el desarrollo de soluciones que no compartan información con el exterior se ha convertido en una nueva línea de desarrollo: modelos de código abierto que nos permitan disfrutar de las ventajas de la IA generativa, pero sin tener que preocuparnos por el uso posterior que reciben nuestros datos porque en ningún momento van a filtrarse a red.

- **BARD:** gratis
- **Llama:** gratis

- **Falcon: gratis**

Estos modelos son extremadamente exclusivos. Necesitan ser entrenados con grandes cantidades de datos y usando costosas computadoras, por lo que sólo lo pueden hacer grandes corporaciones como Google o OpenAI.

Las grandes tecnológicas que desarrollan la más moderna tecnología relacionada a los modelos amplios de lenguaje restringen su uso y no revelan información acerca de cómo fueron creados. Este modelo de secretismo y avaricia en la alta tecnología es lo que se quiere evitar con el lanzamiento libre de BLOOM.

El grupo de investigadores desarrolló una estructura de control de datos, especialmente pensada para LLMs, que debiera clarificar cuál es la información que está siendo utilizada y a quién pertenece y obtuvo diferentes conjuntos de datos que no están disponibles online. También creó una Licencia de IA Responsable, para evitar que sea usado con malas intenciones, pero la realidad es que no hay forma de evitar que esto suceda. “BigScience ha hecho un espectacular trabajo de construir una comunidad alrededor de BLOOM, y su interés por el problema ético y de control desde su comienzo ha sido inteligente”, sostiene Percy Liang, director del Centro de Investigación de Modelos de Lenguaje De Stanford. “Sin embargo, esto no va a cambiar mucho la esencia de los LLMs. OpenAI, Google y Microsoft siguen avanzando a pasos agigantados”, afirma Liang. Finalmente BLOOM es también un modelo amplio de lenguaje, presenta los mismos riesgos y fallos que los otros.

La especificidad de estos algoritmos es que evolucionan, se vuelven más expertos en función de la afluencia y el análisis de datos. En otras palabras, aprenden y mejoran por sí mismos a medida que procesan más datos, ajustando sus parámetros para arrojar mejores previsiones. Por eso la escala es importante para el desarrollo y el uso de la IA. Datos y algoritmos son complementarios e igualmente indispensables para que la IA funcione. En la medida en que estos modelos se vuelven mejores con el uso, podemos pensarlos como medios de producción que se aprecian con el tiempo, lo cual los hace únicos, ya que el resto de los medios de producción se deprecian con el uso.

Es tan variada su aplicación que se ha comenzado a hablar de la IA como una tecnología de propósito general, o incluso de un nuevo método para inventar, en la medida en que permite automatizar descubrimientos y expandir el tipo de problemas que se pueden estudiar con big data. En última instancia, el objetivo de las grandes empresas que están detrás de estos modelos es producir una tecnología que no necesite procesos de adaptación, que pueda ser adoptada directamente para los entornos más diversos. Ergo, que sus clientes sean potencialmente todas las organizaciones y personas del mundo.

En esa búsqueda, aparece un tipo específico de modelo de deep learning, los Large Language Models (LLM, modelos de lenguaje de gran escala). ChatGPT está basado en el LLM más grande del mundo. Los LLM predicen qué palabra viene a continuación en una secuencia. Como explica Google, “los LLM generan nuevas combinaciones de texto en forma de lenguaje natural. E incluso podemos construir modelos lingüísticos para generar otros tipos de resultados, como nuevas imágenes, audio y también video”. En una entrevista, un empleado de Google los definió como “agénticos” porque el agente inteligente –el programa informático– interactúa con el entorno y aprende a actuar en él. Es una herramienta poderosa porque, como todo modelo de deep learning, los LLM mejoran cuantos más datos procesan, con lo que en cierta medida se externaliza parte de su mejora a usuarios o clientes. Es decir, parte del trabajo que produce la IA –que no es remunerado– es realizado por quienes la consumen. Por ejemplo, cada vez que le hacemos una pregunta o pedimos algo a ChatGPT, se producen más datos que pueden ser usados para mejorarlo.

Conclusión: los Grandes Modelos de Lenguaje (LLMs) están ganando popularidad en el campo de la Inteligencia Artificial, generando un impacto significativo en la agenda internacional. Su desempeño excepcional ha llevado a la necesidad de regulaciones y ha afectado a grandes empresas, como Google y Microsoft. Las LLMs se consideran compañeros esenciales en diversas tareas, potenciando actividades en marketing, programación, toma de decisiones, investigación y escritura, en lugar de reemplazar a los humanos. Este avance tecnológico ha generado una nueva carrera en el campo de la IA.

Sistema multiagente:

Problemática a resolver: Pensé en un local de electrodomésticos en el cual una cadena de suministro global enfrenta desafíos significativos en términos de gestión de inventario. La cadena de suministro experimenta frecuentes interrupciones debido a cambios repentinos en la demanda de productos, variaciones en los plazos de entrega de proveedores y dificultades en la previsión precisa de la demanda. Esto resulta en exceso de inventario, pérdida de ventas por falta de productos y costos logísticos innecesarios.

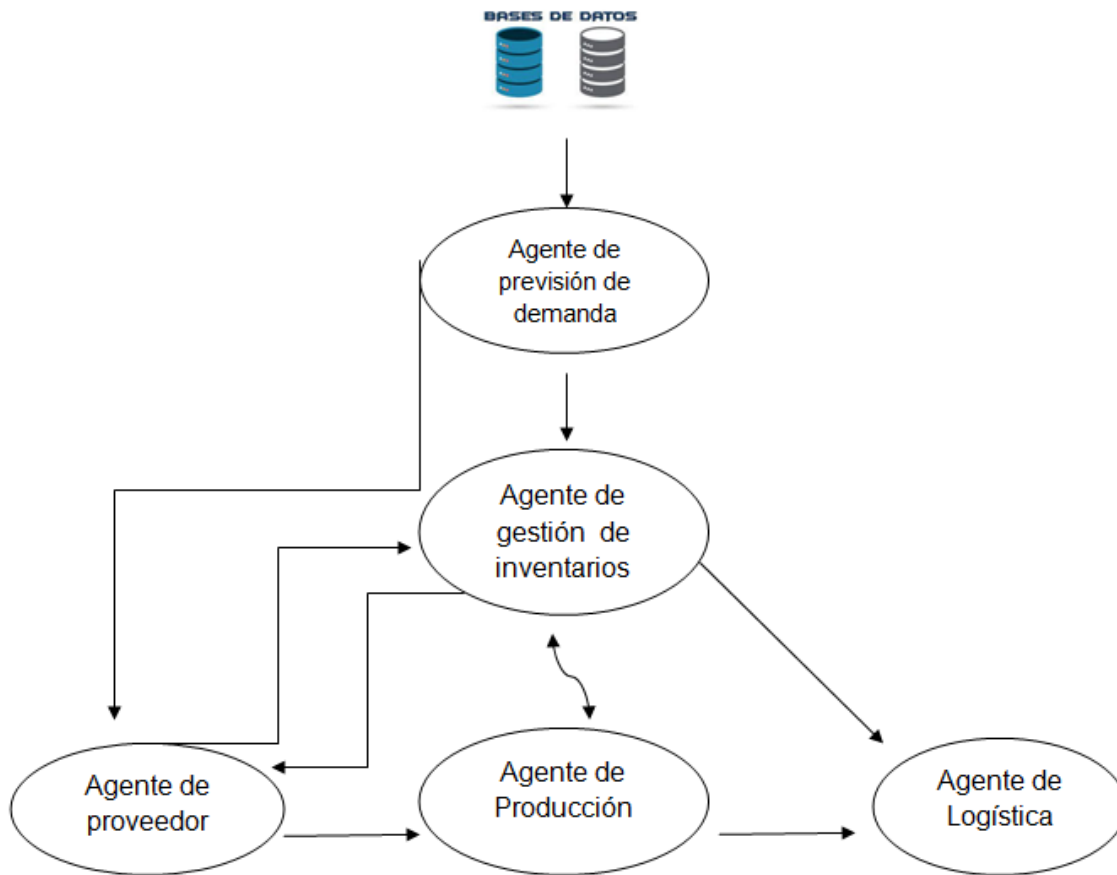
Solución propuesta:

Desarrollar un sistema multiagente que permita una gestión dinámica del inventario, ajustándose en tiempo real a los cambios en la demanda y en las condiciones de la cadena de suministro.

Agentes involucrados en la tarea:

- **Agente de Previsión de Demanda:** utiliza algoritmos avanzados y datos históricos para prever la demanda futura de productos. Proporciona proyecciones actualizadas a intervalos regulares.
- **Agente de Gestión de Inventarios:** monitoriza constantemente los niveles de inventario y ajusta las cantidades en función de las previsiones de demanda y de las variaciones en los plazos de entrega de los proveedores.
- **Agente de Proveedor:** informa sobre los plazos de entrega estimados y cambios en la disponibilidad de productos. Recibe actualizaciones de los niveles de inventario y las proyecciones de demanda.
- **Agente de Producción:** coordina con el Agente de Gestión de Inventarios para ajustar la producción según las variaciones en la demanda y el suministro de materias primas.
- **Agente de Logística:** coordinador de la distribución y transporte de productos. Ajusta las rutas y los tiempos de entrega según las actualizaciones en tiempo real de la demanda y la disponibilidad de inventario.

ESQUEMA DE SISTEMA MULTIAGENTE



En el sistema multiagente , varios agentes interactuarían entre sí para lograr una gestión dinámica del inventario. Las interacciones principales ocurren entre los agentes:

Agente de Previsión de Demanda y Agente de Gestión de Inventarios:

- El Agente de Previsión de Demanda proporciona proyecciones actualizadas de la demanda a intervalos regulares al Agente de Gestión de Inventarios.
- El Agente de Gestión de Inventarios utiliza estas proyecciones para ajustar los niveles de inventario y gestionar eficientemente los pedidos de reposición.

Agente de Gestión de Inventarios y Agente de Proveedor:

- El Agente de Gestión de Inventarios monitoriza constantemente los niveles de inventario y, en función de las proyecciones de demanda y las variaciones en los plazos de entrega, puede enviar solicitudes al Agente de Proveedor.
- El Agente de Proveedor responde proporcionando información sobre los plazos de entrega estimados y cambios en la disponibilidad de productos.

Agente de Gestión de Inventarios y Agente de Producción:

- El Agente de Gestión de Inventarios coordina con el Agente de Producción para ajustar la producción según las variaciones en la demanda y el suministro de materias primas.
- El Agente de Producción actualiza sobre la capacidad de producción y cualquier cambio en la programación.

Agente de Logística y Agente de Gestión de Inventarios:

- El Agente de Logística recibe actualizaciones en tiempo real de la demanda y la disponibilidad de inventario del Agente de Gestión de Inventarios.
- El Agente de Logística ajusta las rutas y los tiempos de entrega según estas actualizaciones.

Estas interacciones permiten que el sistema se adapte dinámicamente a los cambios en la demanda, los plazos de entrega, la disponibilidad de productos y otros factores relevantes, mejorando así la eficiencia de la cadena de suministro. Además, la información intercambiada entre estos agentes contribuye a la toma de decisiones en tiempo real y a la optimización de la gestión del inventario.

Descripción de qué agentes emplean herramientas específicas:

Agente de Previsión de Demanda:

Utiliza algoritmos avanzados para el análisis de datos históricos.

- Accede a bases de datos locales que contienen información histórica de ventas.

Razón:

- La utilización de algoritmos avanzados implica el procesamiento de datos complejos y el análisis de patrones, lo que puede requerir herramientas especializadas.

Agente de Proveedor:

- Puede conectarse a APIs de proveedores externos.
- Accede a información en tiempo real sobre plazos de entrega y disponibilidad de productos.

Razón:

- La conexión a APIs permite obtener información actualizada de los proveedores de manera eficiente y en tiempo real, facilitando la toma de decisiones basada en datos más recientes.

Agente de Logística:

- Accede a información en tiempo real sobre la demanda y la disponibilidad de inventario.

Razón:

- La coordinación en tiempo real requiere acceso a datos actualizados sobre la demanda y la disponibilidad de productos para ajustar las rutas y los tiempos de entrega de manera eficiente.

Agente de Gestión de Inventarios:

- Utiliza sistemas de gestión de inventario y bases de datos locales.

Razón:

- La monitorización constante de los niveles de inventario y la gestión eficiente de pedidos de reposición implican el acceso a información actualizada sobre el inventario y la demanda.

Agente de Producción:

- Coordina con el Agente de Gestión de Inventarios para ajustar la producción según la demanda y el suministro de materias primas.

Razón:

- La coordinación eficiente con el Agente de Gestión de Inventarios puede implicar el uso de herramientas para ajustar los procesos de producción en tiempo real.

Estas herramientas permiten a los agentes del sistema acceder a información crítica, realizar análisis avanzados y coordinar acciones de manera eficiente para abordar los desafíos en la gestión de inventario en tiempo real.

Bibliografía:

<https://www.hostinger.com.ar/tutoriales/modelos-grandes-de-lenguaje-llm>

<https://www.b2chat.io/blog/b2chat/sistemas-multiagente-que-son-como-funcionan/>

<https://www.scalian-spain.es/la-revolucion-de-la-inteligencia-artificial-el-poder-transformador-de-los-modelos-de-lenguaje-a-gran-escala-llm/>