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:

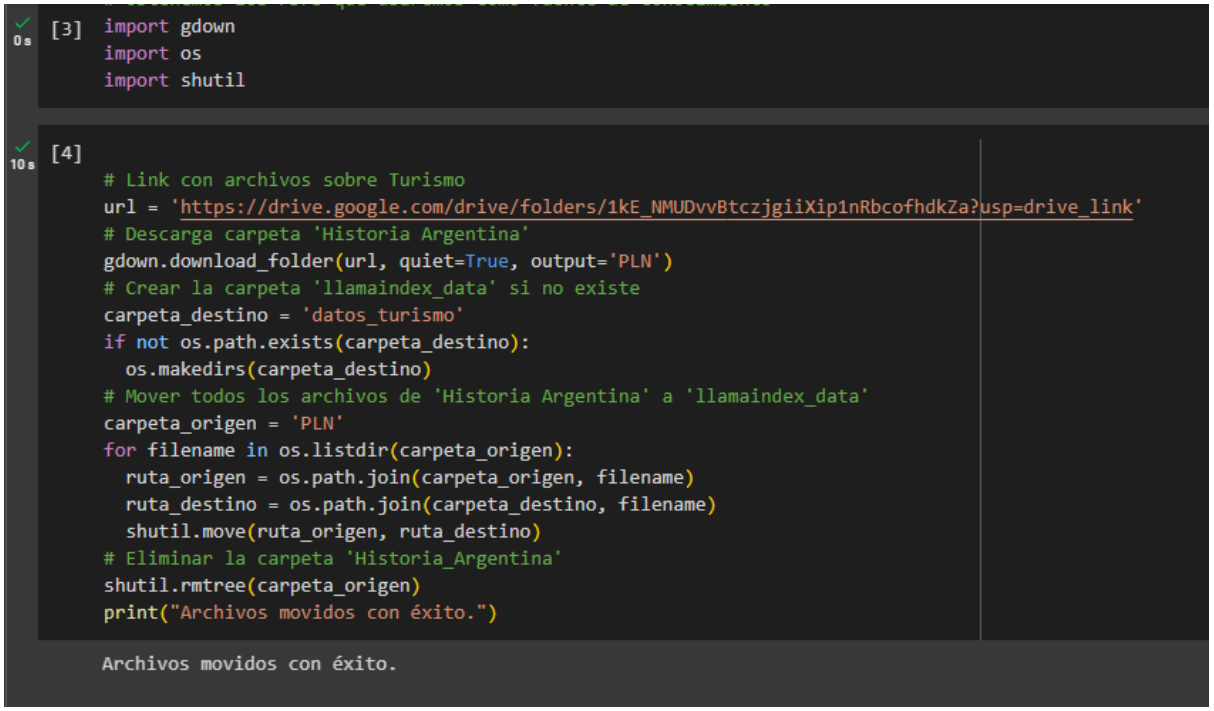
Para empezar,instalamos e importamos las librerías necesarias.

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_NMUDvvBtczjgiiXip1nRbcofhdkZa?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

```python
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

```python
        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 Secre-
tario General de la Organización Mundial del Turismo, Ilmo. Sr. D. Frances-
co 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 comenta-
rios se han incorporado a esta obra: el profesor D. Bernardí Cabrer, el pro-
fesor D. Antonio Juárez, el profesor D. Francisco González-Palazón, Dña.
Adela Moreda, 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.

```python
# 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")
```

```
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, Mourelo 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: subsidios estatales no cambió de manera muy

✓ 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.

```python
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 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.

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

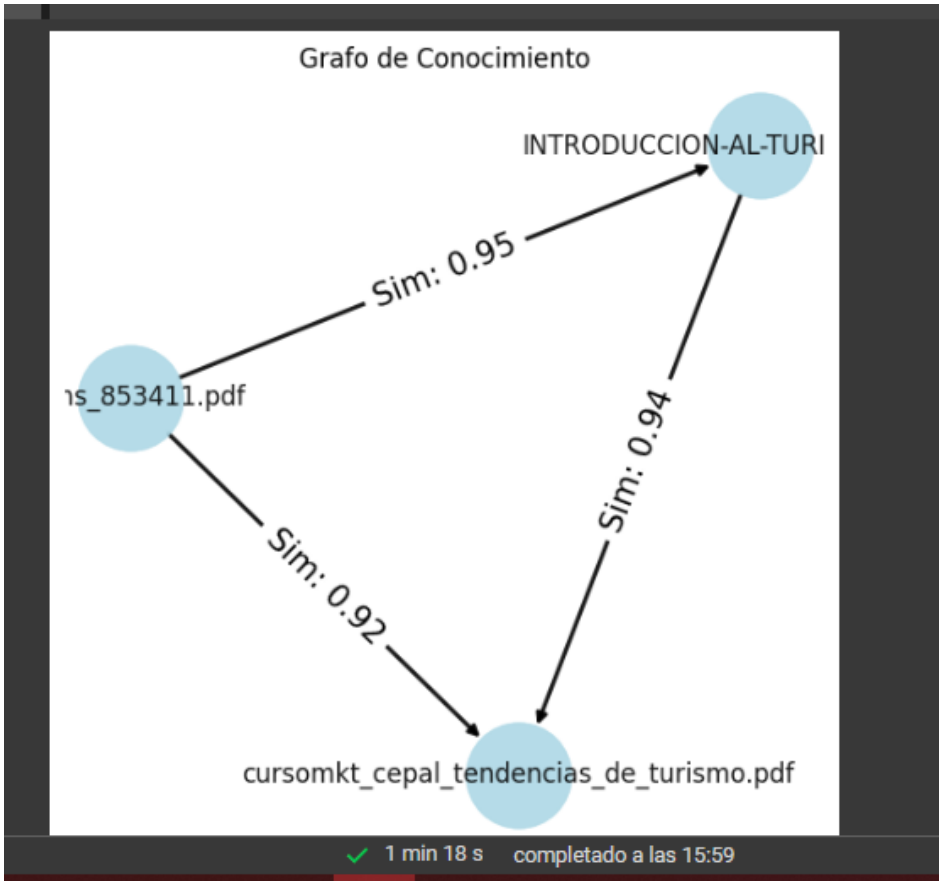
```python
# Lista de embeddings y nombres de documentos
embeddings_documentos = [{'nombre': 'wcms_853411.pdf', 'embedding': [ 5.45836873e-02, 4.73619312e-01, -4.07616436e-01, -6.14194393e-01,
  8.38350892e-01, 4.71917987e-01, 7.19390035e-01, -2.56502777e-01,
 -3.79744351e-01, 3.92292403e-02, 2.40509063e-01, -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, -2.15224206e-01, 6.66436315e-01,
```

```python
[ ]  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.

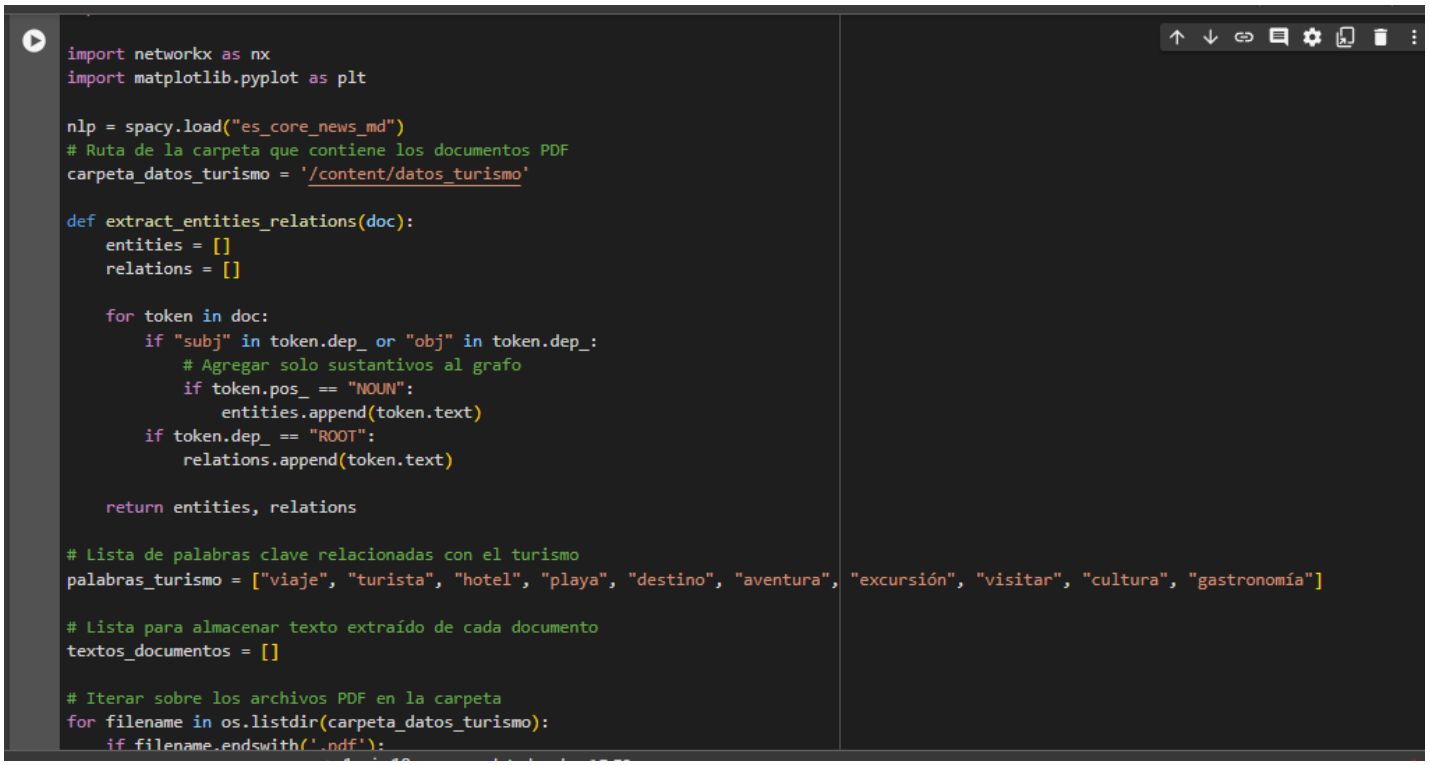
Luego, utilizamos la biblioteca ‘`networkx`’ para construir y visualizar un grafo dirigido de ‘`networkx`’ para trabajar con grafos, `matplotlib.pyplot` para la visualización y ‘`cosine_similarity`’ de scikit-learn para calcular la similitud coseno entre embeddings.

```python
# 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()
```

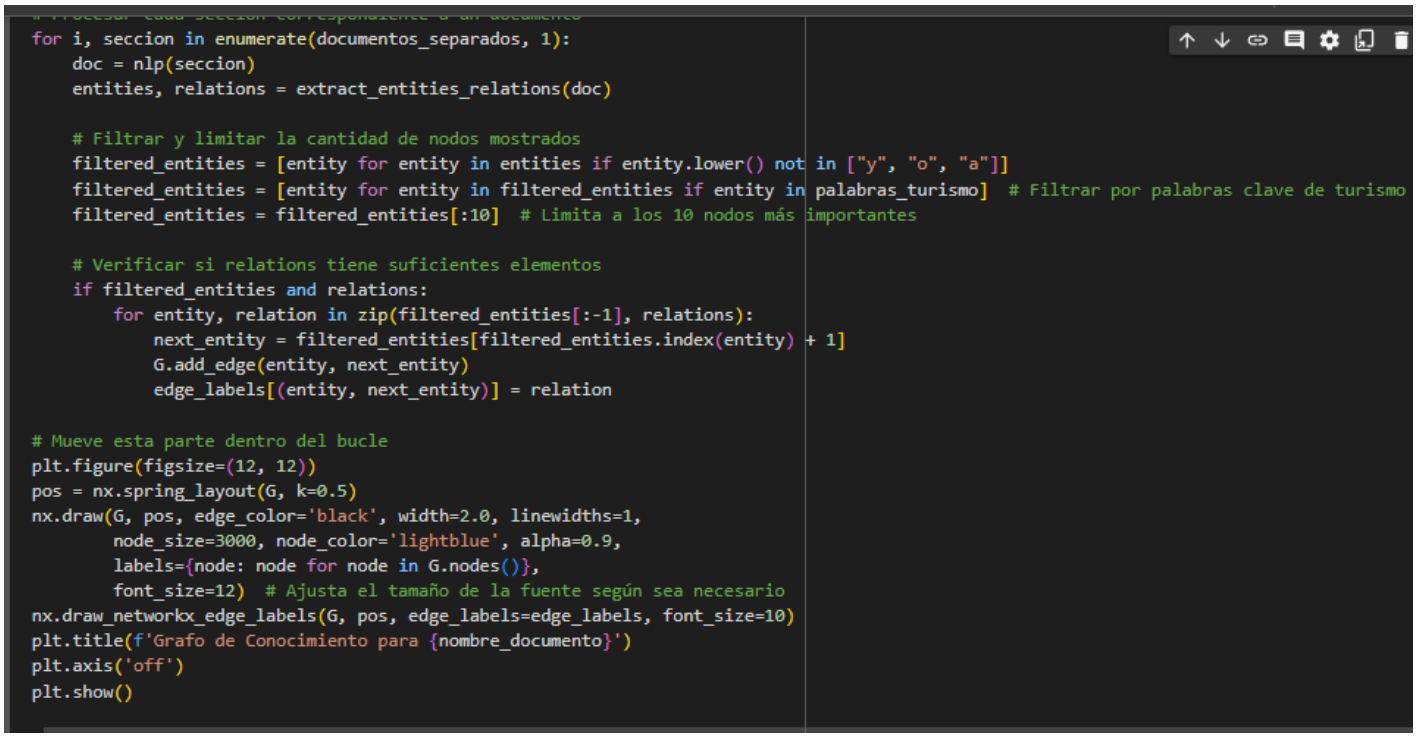


✓ 1 min 18 s completado a las 15:59

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.

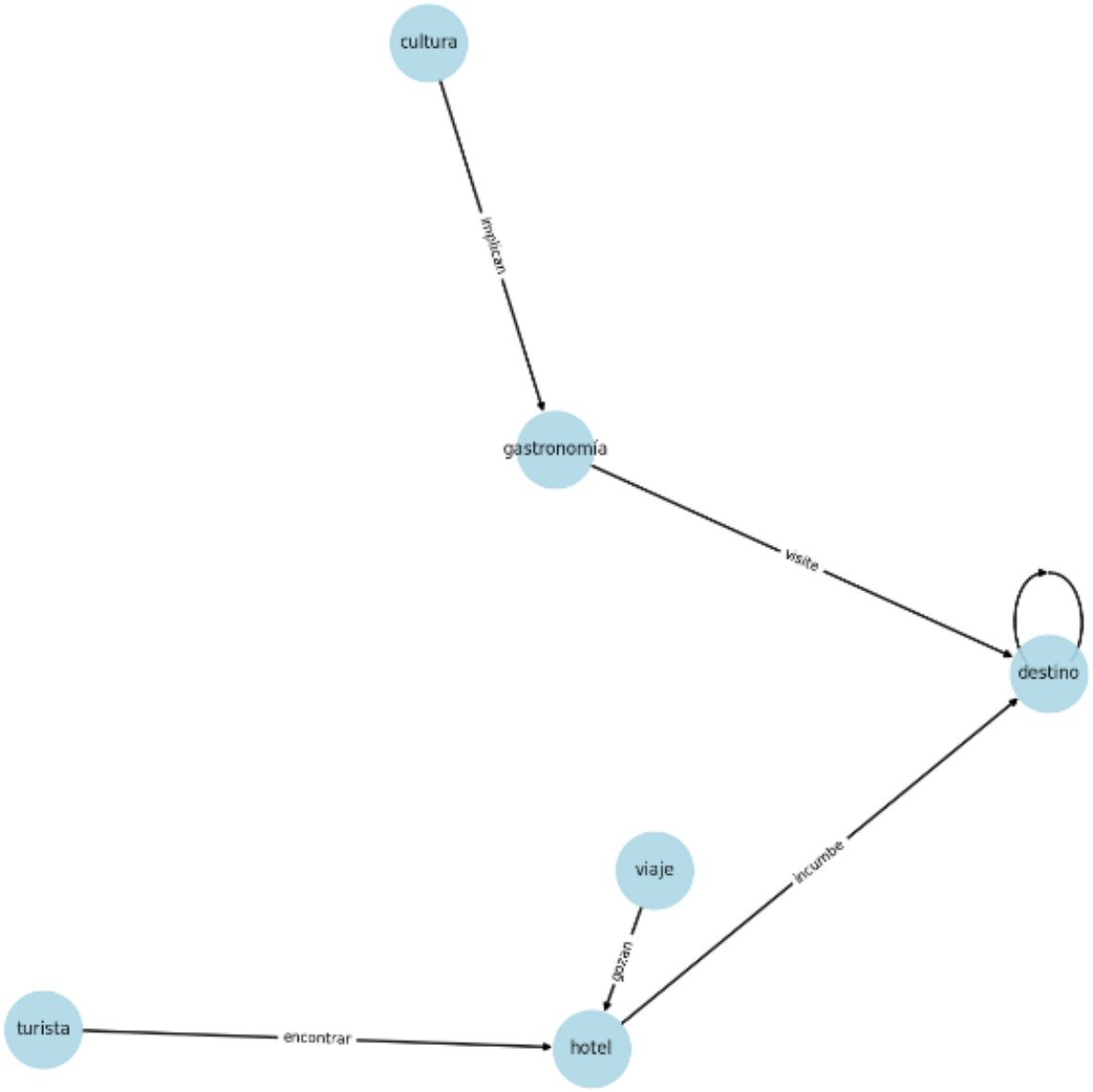


```
import networkx as nx
import matplotlib.pyplot as plt

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'):
```

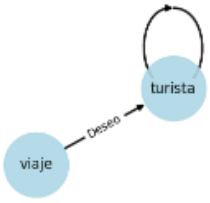


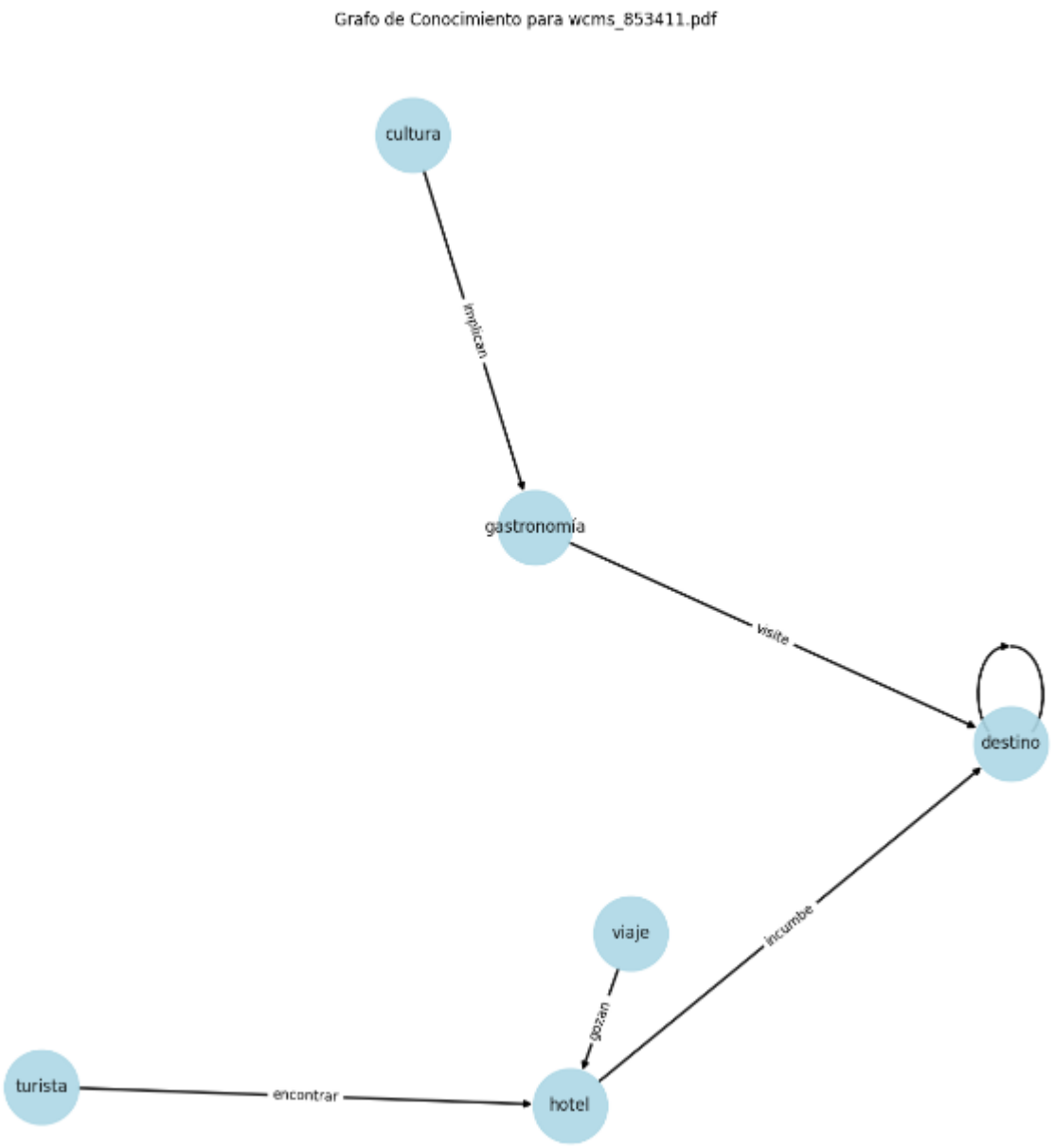
```
for i, seccion in enumerate(documentos_separados, 1):
    doc = nlp(seccion)
    entities, relations = extract_entities_relations(doc)

    # 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()
```

Grafo de Conocimiento para wcms_853411.pdf



Grafo de Conocimiento para INTRODUCCION-AL-TURISMO-OMT.pdf



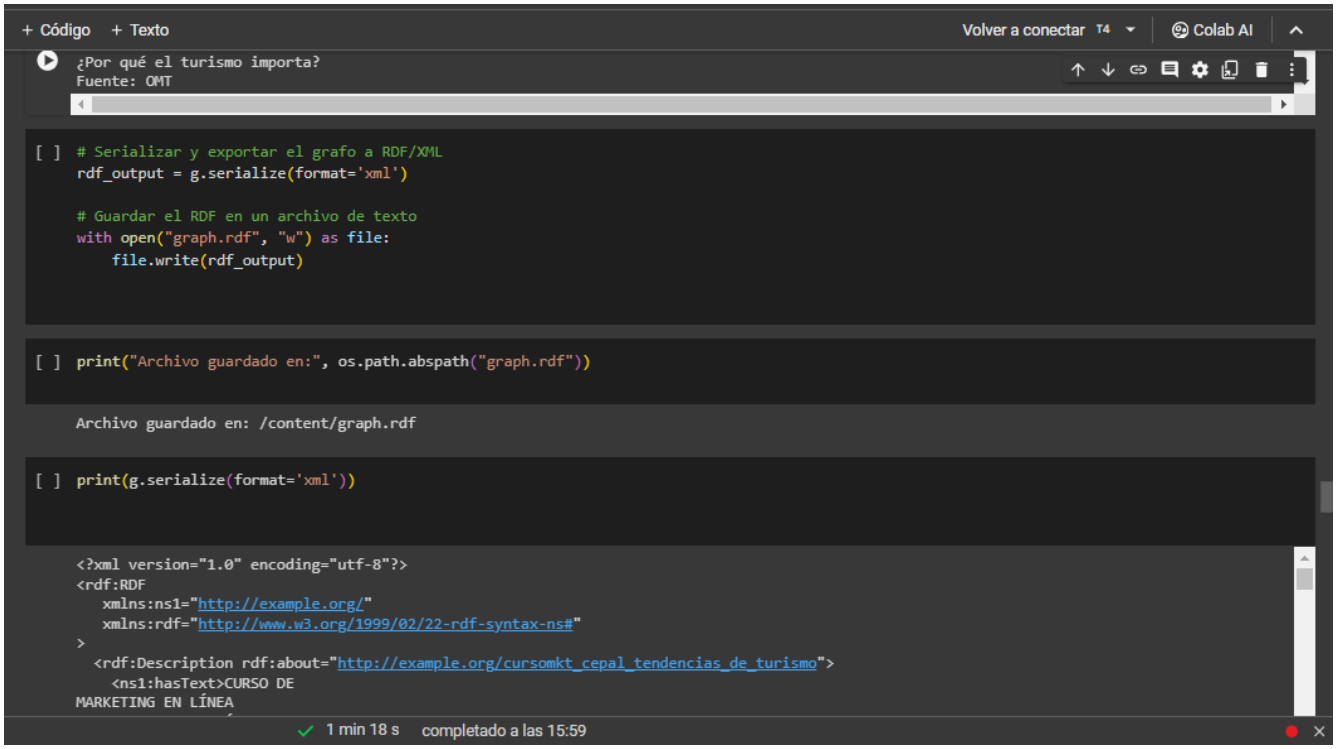


Creamos un grafo RDF que representa la información de texto de los documentos PDF relacionados con el turismo en la carpeta especificada. Cada documento PDF se identifica con un URI único, y el texto del PDF se almacena como un literal asociado a la propiedad `hasText` en el grafo RDF

```python
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)
```

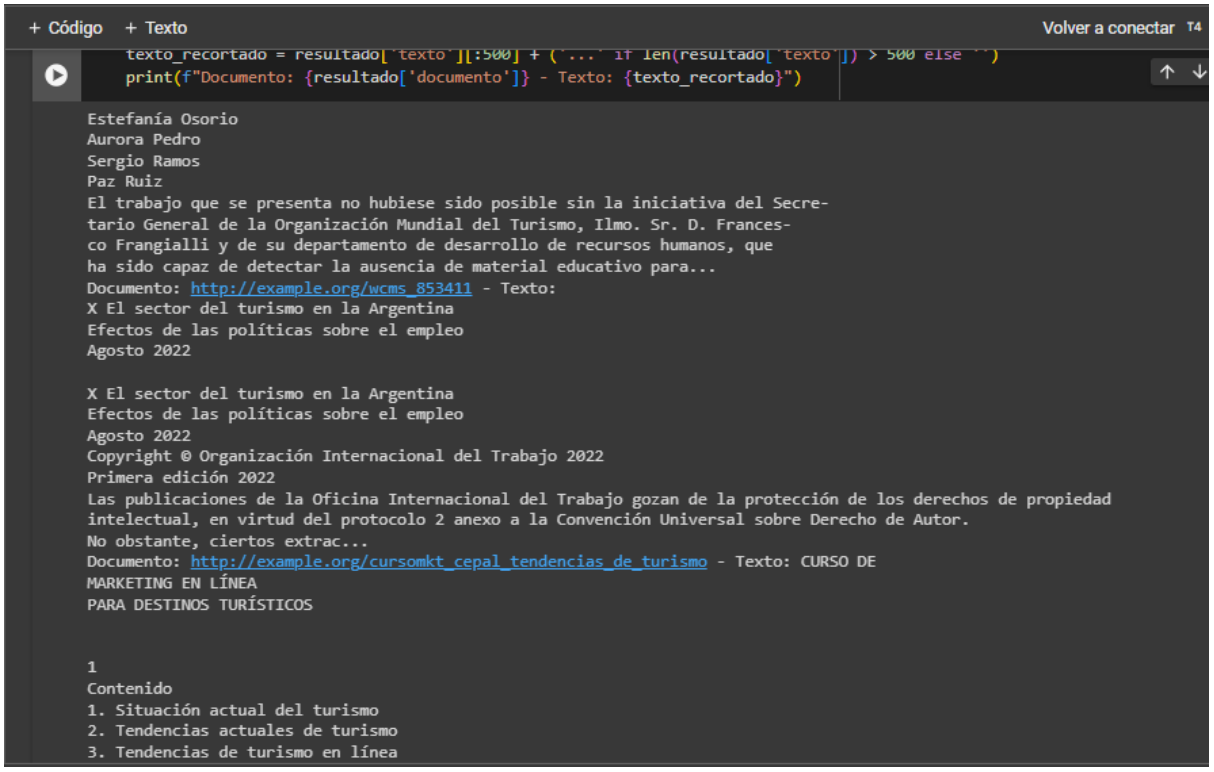
Lo guardamos en un txt

```
<?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)
```



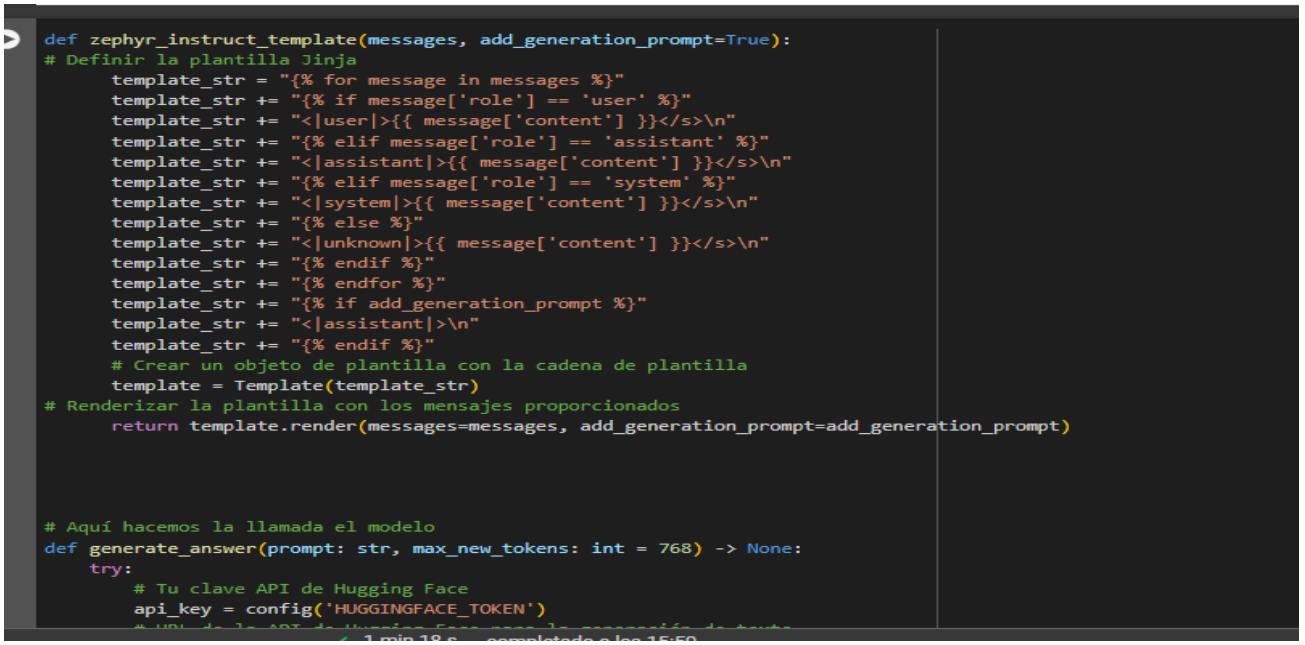
Hacemos una consulta sobre la base de datos de grafos, se ejecuta la consulta SPARQL sobre el grafo RDF creado anteriormente utilizando la función `query` de la biblioteca `rdflib`.

```
# 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}")
```



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.



```python
# Aquí hacemos la llamada el 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

```python
[ ]  # 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'] = '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'

    # 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 aerolieas 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.

```python
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 aerolieas 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)

    # Imprimimos 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 aerolieas 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.
------------------------------------------------------
```

Se me ocurrió implementar una interacción de chat donde el usuario ingresa preguntas, el chatbot genera respuestas y la conversación continúa hasta que el usuario decide salir ingresando "exit".

Interacción con el usuario final

```
[ ]  #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

         # Preparar el prompt para la pregunta del usuario
         prompt_usuario = f"Usuario: {pregunta_usuario}\nChatbot: {generate_answer}\n"

         # Generar y mostrar la respuesta del chatbot
         respuesta_generada_usuario = generate_answer(prompt_usuario)
         print("Chatbot:", respuesta_generada_usuario)
```

Resultado final:

```
Interacción con el usuario:
Usuario: En que puedo ir de Bs As al sur de Argentina?
Chatbot: Al sur de Argentina, puedes ir a varios destinos, algunos de los más populares son:

1. Bariloche: Una ciudad de montaña conocida por su belleza natural y sus actividades de invierno como esquí, snowboard y esquí de montaña.

2. Ushuaia: La ciudad más austral del mundo, conocida por su paisaje glacial y sus actividades de aventura como el kayak, el trekking y el esquí de verano.

3. Tierra del Fuego: Un parque nacional que se extiende por la región austral de Argentina y Chile, conocida por su paisaje escénico y su fauna silvestre, como el puma

4. Punta Arenas: Una ciudad chilena ubicada en la región de Magallanes, conocida por su puerto natural y sus actividades de aventura como el kayak, el trekking y el es

5. El Calafate: Una ciudad ubicada en la Patagonia argentina, conocida por su paisaje glacial y sus actividades de aventura como el trekking y el esquí de verano.

Estos destinos son accesibles por vía aérea o terrestre, y ofrecen varias opciones de alojamiento, desde hostales económicos hasta hoteles de lujo. Es recomendable pre
Usuario: con que aerolineas puedo viajar al sur de Argentina?
Chatbot: Hay varias aerolíneas que operan vuelos hacia el sur de Argentina. Algunas de ellas son:

1. Aerolíneas Argentinas: Es la aerolínea más grande del país y opera vuelos a destinos como Bariloche, Ushuaia, Puerto Madryn, Trelew y Comodoro Rivadavia, entre otro

2. Flybondi: Esta aerolínea low-cost opera vuelos a destinos como Bariloche, Puerto Madryn y Trelew, entre otros.

3. Andes Lineas Aéreas: Esta aerolínea opera vuelos a destinos como Bariloche, El Calafate, Trelew y Comodoro Rivadavia, entre otros.

4. LADE: Esta aerolínea es propiedad del gobierno argentino y opera vuelos a destinos como Ushuaia y Puerto San Julián, entre otros.

5. Austral: Esta aerolínea opera vuelos a destinos como Ushuaia, El Calafate, Puerto Madryn y Comodoro Rivadavia, entre otros.

Es recomendable revisar la programación de vuelos y las tarifas de cada aerolínea para elegir la opción que mejor se adapte a sus necesidades y presupuesto.
Usuario: exit
Chat finalizado. Hasta luego.
```

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 facilitarle 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.

La propiedad de los datos se refiere a quién tiene el derecho legal de poseer y controlar el uso de los datos recopilados. En muchos casos, los términos de servicio ambiguos y las políticas de privacidad permiten a las empresas recopilar y utilizar datos de los

usuarios de formas que pueden no ser completamente comprendidas por el individuo. Por otro lado, la privacidad de los datos se refiere a cómo se protegen estos datos de ser accedidos o utilizados sin el consentimiento del propietario. A pesar de las leyes y regulaciones existentes, la rápida evolución de la tecnología a menudo supera la capacidad de las políticas para mantenerse al día, lo que resulta en brechas de seguridad y violaciones de la privacidad. Este problema complejo requiere un enfoque multidisciplinario que combine la ciencia de datos, la ética y el derecho para garantizar que los datos se utilicen de manera responsable y segura.

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

BLOOM

El BLOOM (BigScience Large Open-science Open-access Multilingual Languaje Model), es un modelo de lenguaje multiidioma, creado por el Centro Nacional de Francia para el Desarrollo Científico. Es similar al GPT-3 desarrollado por OpenAI y al LaMDA, desarrollado por Google.

Selecciona el idioma en el que se quiere trabajar y a continuación elegir lo que se quiere hacer. Puede ser: escribir un poema o una receta, traducir o sintetizar un texto, o escribir códigos de programación. Los desarrolladores de IA pueden usar el modelo como base para construir sus propias aplicaciones.

Con 176.000 millones de parámetros (variables que determinan cómo la información que entra se convierte en la deseada información que sale), BLOOM es más grande que GPT-3, que tiene 175.000 millones de parámetros. Y ofrece niveles similares de certeza y toxicidad (prejuicios) que los otros modelos mencionados. Y a diferencia de estos modelos, BLOOM está disponible en español y árabe.

Son unos de los ámbitos preferidos por los investigadores en inteligencia artificial. Modelos poderosos como GPT-3 y LaMDA, que producen textos como si fueran escritos por personas, tienen gran potencial de cambiar el modo en que procesamos la información online. Pueden ser usados como chatbots o para buscar información,

moderar contenido, hacer reseñas de libros o generar páginas enteras de texto a partir de ciertas indicaciones.

Con suma facilidad estos modelos empiezan a producir contenido dañino.

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.

Meta recientemente anunció su propio modelo amplio de lenguaje, llamado Open Pretrained Transformer (OPT-175B), y puso a disposición su código y un libro detallando cómo fue entrenado el modelo. El OPT sólo está disponible bajo solicitud y su uso está limitado a investigaciones. En el caso de Huggins Face, no sólo es de libre acceso, sino que además están grabadas y disponibles las reuniones que se hicieron mientras se desarrolló el proceso y pueden ser descargadas libremente.

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. OpenAI no ha hecho público su GPT-3, porque según ellos el lenguaje racista y sexista que evidencia lo hace muy peligroso para ser usado públicamente.

Margaret Mitchell, investigadora de Hugging Face, sostiene que BLOOM también tiene sus defectos y prejuicios, pero al ser un modelo abierto, los usuarios podrán colaborar en sus virtudes y señalar sus defectos.

La mayor contribución de BigScience a la inteligencia artificial puede ir mucho más allá de BLOOM, que puede ser solo el comienzo de una forma de cooperación internacional en investigación de inteligencia artificial.

En la actualidad, la técnica más avanzada para procesar datos es un subconjunto de algoritmos de IA, una clase de machine learning llamada aprendizaje profundo (deep learning) o redes neuronales profundas (deep neural networks). Esta es la tecnología de IA que más creció en solicitudes de patentes entre 2012 y 2016, y su desempeño es mejor que el de cualquier otro modelo de IA. Los modelos de deep learning están constituidos por muchas capas que forman redes jerárquicas para aprender características y clasificar identificando patrones.

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 estrictos 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 están apoderándose de toda la popularidad de la Inteligencia Artificial y lo tienen justificado; son realmente grandiosas en sus tareas y han logrado traer a la agenda de los organismos internacionales la importancia de regular este tipo de tecnologías, su importancia, riesgos e impacto que tendrá en nuestra sociedad (global), incluyendo el plano económico y laboral.Las LLMS han pateado el tablero a las grandes compañías, creando una nueva carrera en IA, el propio Google vio amenazado su negocio como motor de búsqueda, Microsoft trazo alianzas estratégicas con OpenAI y reflotó a Bing agregando funciones de Chat con IA y dando acceso a LLMs desde su servicio en la nube Azure.Las personas que están trabajando con esta tecnología creen que las LLMs se han convertido en compañeros indispensables para casi cualquier tareas, potenciando nuestras tareas, no para reemplazarnos si no para aumentarnos (en marketing, programación, toma de decisiones, investigación, escritura…)

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 cambios repentinos en la demanda de productos, variaciones en los plazos de entrega de proveedores 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.

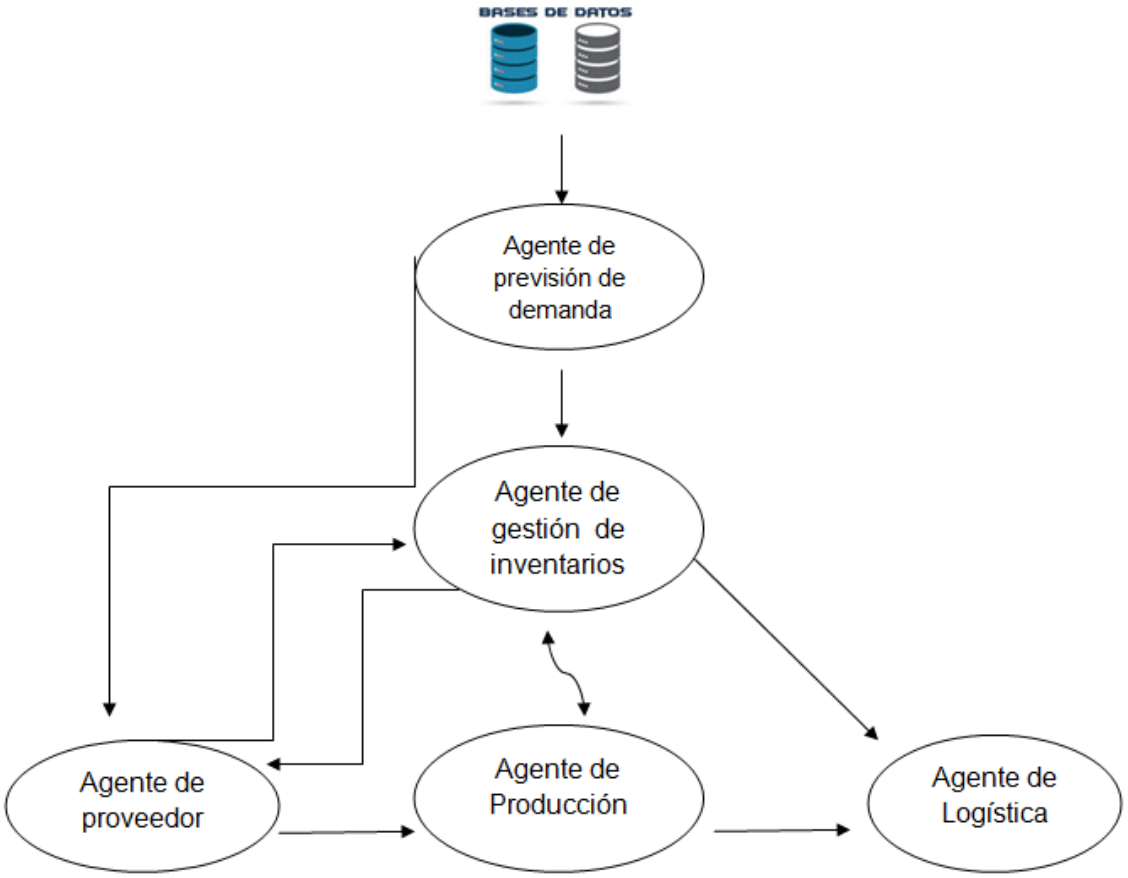
- Agente de Gestión de Inventarios: monitoriza constantemente los niveles de inventario y ajusta las cantidades en función de las previsiones de demanda y de las variaciones en los plazos de entrega de los proveedores.

- 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 interactuarian 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.

Bibliografia:

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/