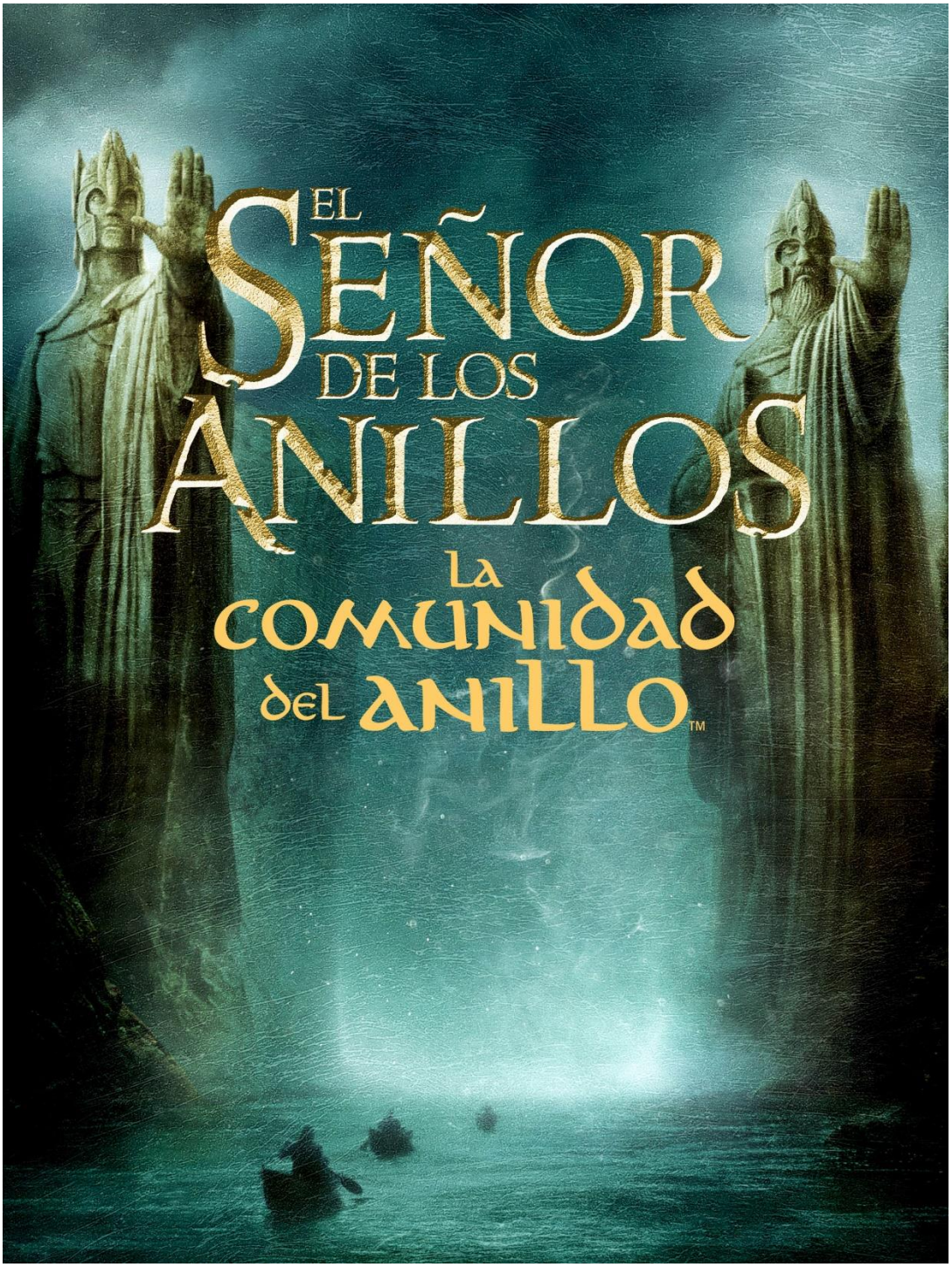
MEMORIA PRÁCTICA DE TEXT MINING:

EL SEÑOR DE LOS ANILLOS



OBJETIVO DE LA PRÁCTICA

El propósito de esta práctica es aplicar los conceptos que se han aprendido en la segunda parte del curso sobre Minería de Texto, en particular, la utilización de redes semánticas y resúmenes de texto automáticos. Para ello, se va a realizar un resumen por extracción del libro "La comunidad del anillo", que es la primera parte de la trilogía "El señor de los anillos" (gran libro). En otras palabras, se va a utilizar la tecnología de procesamiento de lenguaje natural para identificar las partes más importantes del libro y resumirlas de forma automática.

PROCESOS REALIZADOS Y EXPLICACIÓN DEL CÓDIGO

En el código de Python de la práctica se realizan los siguientes pasos:

Lectura del pdf

Primero, importo las bibliotecas necesarias para trabajar con archivos PDF y expresiones regulares. Utilizo re para trabajar con expresiones regulares y pdfminer para extraer texto de archivos PDF.

Luego, defino la variable *pagina_inicio_capitulos* con el valor 55. Este valor indica desde qué página comenzaré a extraer el texto. Esto se puede modificar para cada libro que se quiera usar el código, dado que al usar la expresión regular para dividir por “Chapters” la usamos a partir del capitulo uno para evitarnos todas las apariciones previas.

A continuación, defino una función llamada *extraer_texto_por_capitulos* que recibe un argumento, *archivo_pdf*. Esta función será utilizada para extraer el texto de un archivo PDF y dividirlo en capítulos.

Dentro de la función, abro el archivo PDF en modo binario y comienzo a iterar sobre sus páginas usando *extract_pages*. Si la página actual es anterior a la página de inicio de los capítulos, simplemente la ignoro y continuo con la siguiente página.

Cuando llego a una página dentro del rango de capítulos, examino sus elementos. Si encuentro un elemento que es un contenedor de texto, extraigo su texto y lo añado a la variable *texto_completo*.

Una vez que he recorrido todas las páginas, reemplazo cualquier cantidad de espacios en blanco consecutivos con un solo espacio en *texto_completo*.

Luego, utilizo una expresión regular para dividir *texto_completo* en capítulos. La expresión regular busca la palabra "Chapter" seguida de uno o más dígitos. Ignoro el

primer elemento de la lista resultante, ya que es todo el contenido previo al primer capítulo. Por esa razón cojo la lista de capítulos desde el primer elemento hasta el último.

Finalmente, retorno la lista de capítulos.

Después de definir la función, especifico el nombre del archivo PDF que quiero procesar y llamo a la función *extraer_texto_por_capitulos* con ese archivo. Imprimo la cantidad de capítulos que encontré en el archivo PDF para comprobar que está bien el código.

Creación del grafo usando sinónimos

En este paso primero importamos las librerías necesarias para esta parte del código.

A continuación se utiliza la biblioteca Spacy para procesar un texto en inglés y generar un objeto que contiene información lingüística y semántica del texto. Además, se utiliza la biblioteca NetworkX para generar una red semántica vacía que servirá para representar las relaciones semánticas entre las palabras del texto.

He creado un diccionario para comprobar que lo que iba haciendo estaba bien. Este paso era “doble” trabajo al tener que almacenar los valores en el diccionario a la vez que en el grafo, pero me servía de comprobación.

He cogido todas las palabras del texto que son nombres, verbos, adjetivos o adverbios. He creado una lista con los sinónimos de cada token. El código en esa parte recibe un token, saca todos los sinónimos con wordnet.synsets, los lematiza y los agrega a la lista siempre que no sea la palabra original(el token que recibe por entrada).

A continuación se recorre la lista de sinónimos y vemos si hay alguno en el diccionario creado. Si en la lista de sinónimos aparece una palabra que es clave en algún elemento del diccionario para el bucle. Al pasar esto se agrega la la lista de *new_concepts* el sinónimo encontrado que es la palabra ya presente en el diccionario. Si el token no se encuentra entre los atributos de la clave (sinónimo encontrado) se agrega como atributo. Realizamos lo mismo en el grafo, es decir, si no el token no está como atributo del nodo, se agrega como tal.

Si no se encuentra ningún sinónimo agregamos al diccionario como clave y como valor el token. También agregamos el nodo al grafo y agregamos el token a la lista de *new_concepts*.

Finalmente recorremos las listas para la creación de las aristas entre nodos y creamos el grafo usando la biblioteca de matplotlib.

Este código debería sacar casi todos los sinónimos. El principal problema radica en que recorro solo las claves del diccionario para ver si el sinónimo está presente en las claves. Si no está presente en las claves pero si en los atributos no lo encontrará y lo

agregará como nuevo concepto. Confío en la biblioteca de wordnet para que de todos los sinónimos posibles y en este caso nos dejemos los mínimos posibles por el camino.

Creación de temas y topics

En esta parte hemos usado el modelo *Latent Dirichlet Allocation* (LDA) que es un algoritmo de aprendizaje no supervisado que identifica temas latentes en una colección de documentos al buscar patrones de co-ocurrencia de palabras. Estos temas pueden ser utilizados para diversas aplicaciones, como clasificación de documentos, resumen de texto y análisis de sentimiento.

Primero, creo listas vacías para almacenar los nodos de cada capítulo y los nodos únicos en todas las redes. Luego, itero a través de las redes de cada capítulo, extrayendo los nodos y agregándolos a estas listas.

Después de extraer todos los nodos, creo un objeto Dictionary de Gensim utilizando los nodos de cada capítulo. Este diccionario se utiliza más adelante para convertir los nodos en una representación de bolsa de palabras.

A continuación, itero a través de los nodos de cada capítulo y calculo el número de temas como el 1% de la cantidad de nodos en el capítulo (ajustado por el factor de compresión). Me aseguro de que el número de temas sea al menos 1.

Para cada capítulo, creo un corpus utilizando la representación de bolsa de palabras de los nodos. Luego, entreno un modelo LDA con este corpus y el diccionario creado anteriormente utilizando el número de temas calculado en el paso anterior y 15 iteraciones(passes) para el entrenamiento.

Escojo 5 palabras por tema, puesto que no había nada en las instrucciones y me pareció un número adecuado.

Finalmente, imprimo los temas generados por el modelo LDA para cada capítulo, mostrando las palabras clave y sus probabilidades asociadas.

Unión de las redes semánticas para obtener la red global

Posteriormente he unido todo lo anterior dentro de la función *generar_red_semantica* para poder generar una red semántica de todos los capítulos uno a uno. Antes lo tenía fuera de la función para ir metiendo el la variable *text* un capítulo cualquiera e ir haciendo pruebas pequeñas para ver que estaba ejecutando todo correctamente.

A continuación creo "*redes_capitulos*" para crear la red semántica global a través de un bucle con el que voy generando cada red de un capítulo y agregándola a la lista. Luego

con la función de “nx.compose” voy agregando las redes de cada capitulo una a una para finalmente obtener la red global.

Creación del factor de compresión por nodos

En esta parte importo la biblioteca necesaria. Recorro la variable capítulos y por cada capitulo voy sacando todas las frases para guardar todas las frases del texto en una lista.

Hay una parte del código entre comillas que imprime todas las frases que tienen menos de 15 caracteres. Este número lo puse al azar para coger frases cortas y ver que tipos de frases son y así ver si correspondía la pequeña diferencia que hay de frases entre el capitulo original y cuando se procesan nodos (las frases que no contienen ningún nodo son las que no aparecen). Al ser sólo un código de comprobación se presenta sin opción de ejecutar.

Posteriormente creo el diccionario nodos que contiene cada nodo de la red global y como valor su número de aristas. Ordeno el diccionario de mayor a menor y creo el factor de compresión n. Obtengo los nodos correspondientes al factor de compresión elegido y creo la variable resumen. Para cada frase del texto si el nodo está en la frase y la frase no está en el resumen agrego a la lista la frase correspondiente.

Así obtendríamos un resumen del texto en función de los nodos más importantes.

Creación del factor de compresión por frases

Para crear esta parte primero, creo un diccionario vacío llamado "*dic_frases*" que almacenará las frases del resumen y la suma de las aristas de los nodos en cada frase. Itero a través de las frases en el resumen y para cada frase inicializo una variable "suma" en cero. Luego, itero a través de todos los nodos y si un nodo está presente en la frase, sumo sus aristas al valor de "suma". Almaceno la frase y la suma de aristas de los nodos en el diccionario "*dic_frases*" con el índice de la frase como clave.

Después, ordeno las frases del resumen en función de la suma de aristas de los nodos en cada frase, de mayor a menor. Establezco un factor de compresión adicional del 10% (0.1 en este caso) y calculo el número de frases que quiero seleccionar en función de este factor de compresión.

Selecciono los índices de las primeras "n" frases ordenadas por la suma de grados de nodos y los almaceno en la lista "*top_indices*". Ordeno esta lista de índices para mantener el orden original de las frases en el resumen.

Finalmente, obtengo las frases correspondientes a los índices ordenados y las almaceno en la lista "*top_frases*". Estas frases son las más importantes y representan el resumen del texto original.

Creación del pdf con el resumen del libro

Primero, importo las bibliotecas y módulos necesarios. Luego, creo un nuevo archivo PDF llamado "Resumen_La_comunidad_del_anillo.pdf" con el tamaño de página "letter" utilizando la clase *SimpleDocTemplate*.

Establezco el estilo de los párrafos usando la función *getSampleStyleSheet* y selecciono el estilo "Normal".

Después, creo una lista vacía llamada "*story*" que almacenará los párrafos y espacios que agregaré al PDF. Itero a través de la lista *top_frases* (que contiene las frases más importantes del resumen) y, para cada frase, creo un objeto *Paragraph* con el estilo que definí anteriormente. Además, agrego un objeto *Spacer* para añadir espacio entre las líneas.

Finalmente, construyo el PDF utilizando la función *build* de la clase *SimpleDocTemplate* y lo guardo en el archivo especificado. Imprimo un mensaje de éxito al completar la generación del PDF.

CONCLUSIONES

Teniendo en cuenta el factor de compresión de los nodos, he cogido uno muy pequeño porque sino saldría una gran cantidad de nodos. Para las frases he sacado también un factor de compresión pequeño para obtener un resumen sencillo. Obviamente ambos factores se pueden cambiar en la línea de código correspondiente.

Para los topics también he cogido un factor todavía más pequeño porque al tener alrededor de 800 temas lo mejor era coger un número pequeño para ver si se sacaba algo en claro. Casi todas las palabras tienen pesos cercanos a 0 excepto en algún capítulo, así que yo no he sacado nada en claro personalmente.

Tras realizar la práctica (bastante compleja) he visto la dificultad de sacar una red completa con sinónimos. Aunque el resumen que hace creo que no está nada mal para el código empleado, la conclusión es que todavía no puedo competir con ChatGPT para la generación de resúmenes y redes semánticas.