UNIVERSIDAD NACIONAL DE ROSARIO (UNR)

FACULTAD DE CIENCIAS EXACTAS, INGENIERÍA Y AGRIMENSURA (FCEIA)

TECNICATURA UNIVERSITARIA EN INTELIGENCIA ARTIFICIAL (TUIA)

PROCESAMIENTO DEL LENGUAJE NATURAL (PLN)

Docentes de la cátedra:

Manson, Juan Pablo

Gery, Alan

Trabajo Práctico 1 (TP1)

Estudiantes:

Francisco J. Alomar

Bruno Longo

AÑO 2024 (1er CUATRIMESTRE), Rosario.

INTRODUCCIÓN

El siguiente informe describe el proceso de trabajo realizado durante el TP1 de la asignatura PLN, carrera TUIA, Facultad FCEIA, Universidad UNR. El objetivo principal es construir una base de datos de libros a partir de técnicas de *web scraping*. Luego es utilizada por un programa para que usuarios puedan obtener recomendaciones de lectura en base una descripción *ad hoc* escrita, que deben ingresar por consola. Además, se requiere que el usuario pueda hacer una búsqueda más específica agregando a la descripción ya sea una especificación del género literario o autor. Para ello, diseñamos dos *scripts* en formato .ipynb que justificamos en el informe: básicamente, el primero de ellos se encarga del *web scraping* y construir la base de datos de libros que, luego, es utilizada por el segundo *script* que interactúa con el usuario y hace las recomendaciones de lecturas. Todos los archivos necesarios para ejecutar y acceder al código se encuentran en el siguiente repositorio (incluido este informe). El modo de exposición por el que optamos es más bien descriptivo y de funcionamiento general de los códigos, refiriéndonos a ellos mediante hipervínculos, dado que consideramos que están documentados de modo aceptable. Hacia el final, a modo de conclusión, reflexionamos sobre el trabajo realizado.

1. Script de *web scraping*

El sitio web para el que diseñamos el código fue otorgado por la cátedra en las consignas del TP1: Lecturlandia. La biblioteca que utilizamos para “scrapear” es Selenium y el archivo de código se llama PLN_TP1_web_scraping.ipynb, el cual pasamos a describir. Al comienzo instalamos la biblioteca Selenium y dependencias requeridas que, por defecto, no se encuentra instalada en el entorno de Google Colab. Después definimos dos funciones. La primera configurar_driver() que “setea” el *driver* de Selenium, y la segunda, extraer_datos_libro() que, como su nombre lo indica, desde la url obtenida para cada libro, extrae los datos requeridos para la construcción de parte del *dataset* (autor, género, reseña y la misma url).

El código de *scraping* inicializa un lista vacía (books_data) donde guarda los resultados de la última función descrita, es decir, genera una lista de diccionarios con la información de todos los libros y construye un *data frame* con la biblioteca Pandas. Como puede apreciarse, el código tiene un estructura de try-except-finally. Esto se debe a que al implementar Selenium muchas veces surgen situaciones, por ejemplo, en los tiempos de espera hasta que un elemento es “visible” y falla el algoritmo. La biblioteca ofrece dependencias para contemplar imponderables que pudiéramos hallar al diseñar *scrapings*. En el caso mencionado, la dependencia que permite tratar esta excepción sin que se “rompa” el código es TimeoutException. Hay varias dependencias que importamos de las

cuales pasaremos, someramente, a detallar una de ellas, dado que excede al escrito una descripción más exhaustiva. La búsqueda de elementos necesarios para el *scraping* es mediante el componente “By” de la biblioteca. Esta dependencia permite buscar elementos mediante varias estrategias de localización, de las cuales (al ser varias) sólo mencionamos aquella utilizada: CSS_SELECTOR. Los selectores CSS ayudan a encontrar diversos elementos en una estructura HTML, basándose en distintos atributos, IDs, clases, etc., y combinaciones.

Ahora pasamos a hacer una descripción general del código de *scraping*. Al ingresar a la url indicada (Lecturlandia), el algoritmo busca y hace *click* en “Géneros”. Una vez allí, itera por cada género que cumpla con la condición de tener diez o más pestañas en la parte inferior de la página *web* al acceder a cada uno de ellos. Si la condición se cumple se queda sólo con las primeras diez pestañas (cada una contiene 24 libros). Para pasar de pestaña en pestaña, se hace *click* en el botón de “Siguiente”. De cada pestaña se obtienen las urls con las cuales se construye el *data frame*.

Luego de generar la base de datos, se define una tercera función descripcion_transformer() que sirve para convertir una cadena de texto de dimensión variable en un *embedding* de 512 dimensiones mediante la biblioteca Transformers, Sentence_transformers. Mediante una línea de código se invoca a la función correspondiente y se le agrega al *data frame* una columna con *embeddings* de reseñas por registros correspondiente a cada libro. Finalmente, el *data frame* resultante se guarda en formato .pkl, recomendable para una mejor performance dado que permite optimizar el procesamiento de datos complejos y de alta dimensionalidad como los *embeddings*. De este modo se obtiene el archivo books_data.pkl, que es la base de datos con 1596 libros distintos a utilizar en el próximo algoritmo.

2. Script de recomendación de lecturas.

El *script* PLN_TP1_programa_libros.ipynb inicia con la instalación de bibliotecas necesarias para el manejo de *embeddings* y distancia de Levenshtein, y luego cargando el archivo books_data.pkl que fue generado por el *script* anterior, el cual contiene las columnas *Titles, Authors, Genres, URL, Synopsis* y *Embedded_Synopsis*. Cabe destacar que el archivo del *dataset* está provisto al inicio de este *script* (se descarga directamente desde el repositorio), para evitar la necesidad de generarlo con el código anterior y/o tener que subirlo manualmente al *script*. Solo es necesario correr las celdas en orden.

Luego, se definen las funciones de interfaz para usuario, donde fueron utilizadas las bibliotecas Os, I.python.display, Time y Warnings. En esencia, se implementa un menú que permitirá correr los segmentos del *script* propios a cumplir con los requerimientos del trabajo. En particular las funciones opcion_1(), opcion_2() refieren a la búsqueda por autor y

por género, respectivamente. La búsqueda para ambas opciones es análoga: se normaliza el texto ingresado a través de los *prompts* mostrados en el menú, para luego utilizar la distancia de Levenshtein con un umbral = 5, con el fin de buscar los nombres más cercanos. Para ello, primero son utilizadas las funciones de pre-procesamiento de datos limpiar_y_normalizar_texto() y transformar_str_en_lista(), a fin de manipular cualquier distintos tipos de carácter y "tokenizar" el ingreso de texto.

Para el procesamiento de los datos, son especialmente relevantes las funciones buscar_levenshtein(), description_transformer() y buscar_libro_cercania(). La primera tomará los datos normalizados y cotejará la distancia con las opciones del *dataset* normalizado. En el menú, en caso de no encontrar similitudes, permite realizar una nueva búsqueda; si hay similitudes, son devueltos 3 títulos junto al resto de sus datos.

La función description_transformer() es la encargada de transformar, gracias al modelo pre-entrenado "*distiluse-base-multilingual-cased-v*1", la descripción ingresada en un vector de 512 dimensiones, a fin de que buscar_libro_cercania() compare dicho vector con la columna de sinopsis del mismo formato *Synopsis_Embbebed* del *dataset*. Esto se realiza dentro de la última función mencionada a través de la similaridad del coseno, provista por la librería Sklearn. Finalmente, los 3 vectores más cercanos a la variable vector_descripcion, permiten buscar por su índice el resto de los datos.

CONCLUSIONES

El presente trabajo nos demandó el uso de varios conceptos teóricos y prácticos aprendidos en la materia. En especial, nos llamó la atención la posible conjunción entre *web scraping* y análisis de texto utilizando *embeddings*. Además, nos dedicamos a generar un *dataset* de manera automatizada con ciertos parámetros, notando algunas cuestiones. Primero, que la cantidad de datos puede ser inabarcable y que el resultado del trabajo está fuertemente condicionado por la recolección y armado del *dataset*. Después, al no haber aleatorizado la búsqueda, los géneros no están completamente recorridos, sino que utilizamos los primeros "más grandes", condicionando el producto. Tercero, el análisis de la cercanía de textos nos permitió indagar sobre distintas formas de implementar las búsquedas, y, por último, el uso de *embeddings* es sorprendemente ágil una vez conseguido un buen modelo pre entrenado y un correcto uso de los formatos.