INFORME Y CONCLUSIONES

Trabajo Práctico N.º 1

Técnicas para el Análisis del Lenguaje TF-IDF (Term Frequency-Inverse Document Frequency) Canalización (Pipeline) de NLP

Nombre: Coral Tolazzi
Tema: Técnicas para el Análisis del Lenguaje TF-IDF (Term Frequency-Inverse Document Frequency): Canalización (Pipeline) de NLP
Profesor: Yanina Ximena Scudero
Cuatrimestre y Año: 1 Cuatrimestre del 2025

Instituto tecnológico Beltrán
Procesamiento de Lenguaje Natural

Objetivo del trabajo

Este trabajo práctico tuvo como objetivo implementar un pipeline de procesamiento de lenguaje natural (NLP) para analizar un corpus de textos. A lo largo del ejercicio, está en práctica técnicas fundamentales del análisis textual, como la eliminación de stopwords, la lematización de palabras, el cálculo de la matriz TF-IDF (Term Frequency – Inverse Document Frequency) y la visualización de la frecuencia de términos.

Utilización del corpus

El archivo `CorpusLenguajes.txt` que contiene texto plano, pero incluía código Python, el cual definía una variable llamada `corpus`, que luego de estructurarse, se convierte en una lista de listas de palabras lematizadas. Así que primero se debía ejecutar el archivo para acceder a su contenido.

Entonces se utilizó la función `obtener_corpus`, que:

- Lee el archivo y ejecuta su contenido de forma aislada usando `exec()`.
- Extrae la variable `corpus`, con las oraciones lematizadas como listas de tokens.
- Genera una segunda versión del corpus, uniendo los tokens de cada oración en un solo string (útil para usar con TF-IDF).

De esta forma, se obtuvieron dos representaciones:

- `corpus`: lista de strings, con cada oración como una cadena completa.
- `corpus_lematizado`: lista de listas, manteniendo los tokens individuales por oración.

Tener ambas versiones dio más flexibilidad, ya que algunas técnicas trabajan mejor con texto plano, y otras necesitan un acceso más detallado al nivel de palabra.

Técnicas aplicadas

A partir del corpus procesado, se armó un pipeline de NLP que incluyó las siguientes etapas:

- **Eliminación de stopwords y limpieza**
  - **Función**: `quitarStopwords_eng()`
  - **Qué hace**: Elimina palabras vacías en inglés, signos de puntuación y otros caracteres residuales como `'s`, `--`, etc.
  - **Herramientas usadas**: `stopwords.words("english")` + `string.punctuation` de NLTK.
- **Lematización**
  - **Función**: `lematizar()`
  - **Qué hace**: Reduce cada palabra a su forma raíz (lemmatized), considerando su categoría gramatical.
  - **Herramientas usadas**: `WordNetLemmatizer()` + función auxiliar `get_wordnet_pos()` para mapear etiquetas POS (NLTK).
- **TF-IDF**
  - **Función**: `aplicar_tfidf()`
  - **Qué hace**: Convierte el corpus en una matriz numérica que representa el peso de cada término según su frecuencia relativa en el documento y en todo el corpus.
  - **Herramientas usadas**: `TfidfVectorizer()` de `scikit-learn`.
- **Frecuencia de palabras**
  - **Función**: `obtener_frecuencia()`
  - **Qué hace**: Calcula la frecuencia absoluta de cada palabra lematizada en todo el corpus.
  - **Herramientas usadas**: `FreqDist()` de NLTK.
- **Visualización**
  - **Función**: `graficar_distancia_de_frecuencia()`
  - **Qué hace**: Grafica las 20 palabras más frecuentes del corpus en un gráfico de barras.
  - **Herramientas usadas**: `FreqDist().plot(20)` + `matplotlib.pyplot`.
- **6 palabras más usadas**
  - **Función**: `mostrar_6_palabras_mas_frecuentes()`
  - **Qué hace**: Muestra las 6 palabras con mayor frecuencia en todo el corpus.
  - **Herramientas usadas**: `FreqDist().most_common(6)` de NLTK.
- **Palabra menos frecuente**
  - **Función**: `mostrar_palabra_menos_frecuente()`
  - **Qué hace**: Encuentra la palabra con menor frecuencia absoluta (solo una aparición).
  - **Herramientas usadas**: `FreqDist().most_common()[-1]`.
- **Palabra más repetida en una misma oración**
  - **Función**: `mostrar_palabra_mas_repetida_en_una_misma_oracion()`

- **Qué hace**: Recorre cada oración del corpus y detecta cuál contiene la palabra más repetida dentro de sí.
- **Herramientas usadas**: `FreqDist()` por oración + `most_common(1)` + comparación de repeticiones.

Dificultades encontradas

Una de las principales complicaciones fue entender cómo tratar el corpus que a pesar de su formato `.txt`, contenía código Python. Al principio se intentó procesarlo directamente, lo que daba resultados erróneos, ya que el código también era tomado como contenido del corpus a procesar. Crear la función `obtener_corpus` fue clave para ordenar ese flujo de trabajo. Se empleó la función `exec` para ejecutar el código dentro del corpus. Eso trajo otra complicación, dicha ejecución no retornaba la variable `corpus`, por lo que fue necesario pasar como segundo parámetro el diccionario `contexto` que almacenaría el valor de corpus al finalizar la ejecución. Previo a esto, se usó `globals().copy()` sobre el diccionario porque `exec` necesita acceso a las funciones `lematizar` y `quitarStopwords_eng`.

También surgió una duda al momento de calcular las “palabras más usadas”. Había dos opciones, usar la frecuencia absoluta o los valores del TF-IDF. Como TF-IDF está más orientado a medir relevancia contextual además de la cantidad de apariciones. Entonces, para este análisis tenía más sentido usar la frecuencia absoluta.

Por otro lado, en cuanto al proceso de desarrollo, por más que hubo muchísimos errores, desde problemas al ejecutar el archivo del corpus, hasta confusiones con la estructura de los datos y el uso de funciones del pipeline, honestamente fueron esos errores los que forzaron a profundizar, a investigar más allá del material teórico ya sea con ayuda de Chat GPT o foros cómo Reddit y GitHub, y a entender cómo y por qué funciona cada parte del código.

Conclusión

Este trabajo sirvió muchísimo para afianzar conceptos clave del procesamiento de lenguaje natural. No solo se puede practicar técnicas como la eliminación de *stopwords*, lematización y TF-IDF, sino que además se pudo aprender a adaptar a distintas formas de representar texto. Se aprendieron muchas nuevas funciones aplicables a listas y estructuras complejas en Python. También se mejoró lo de entender cómo rastrear errores si el código no funciona y cómo pensar en estructuras que se adapten a diferentes necesidades. Finalmente, se adaptó y se modularizó el código para que sea más legible y reutilizable, ya que este trabajo es muy valioso para publicarlo en el repositorio de GitHub. Este trabajo no solo es una buena práctica académica, sino que puede resultar útil para cualquier persona que esté comenzando en el mundo del NLP.