Procesamiento del Lenguaje Natural

Trabajo Práctico Individual

Detección de Plagio

Alumno: Diego Peccia

INTRODUCCIÓN

Este documento acompaña al código subido en el siguiente repositorio de Github: https://github.com/dpeccia/nlp2020-tp que corresponde a la solución planteada para el trabajo práctico de *Detección de Plagio* del año 2020 - 1er cuatrimestre, de la materia Procesamiento del Lenguaje Natural, dictada por el profesor Hernán Borré.

Se explica detalladamente esta solución, así como las técnicas utilizadas, y se le da crédito a los autores y páginas consultadas.

HERRAMIENTAS UTILIZADAS

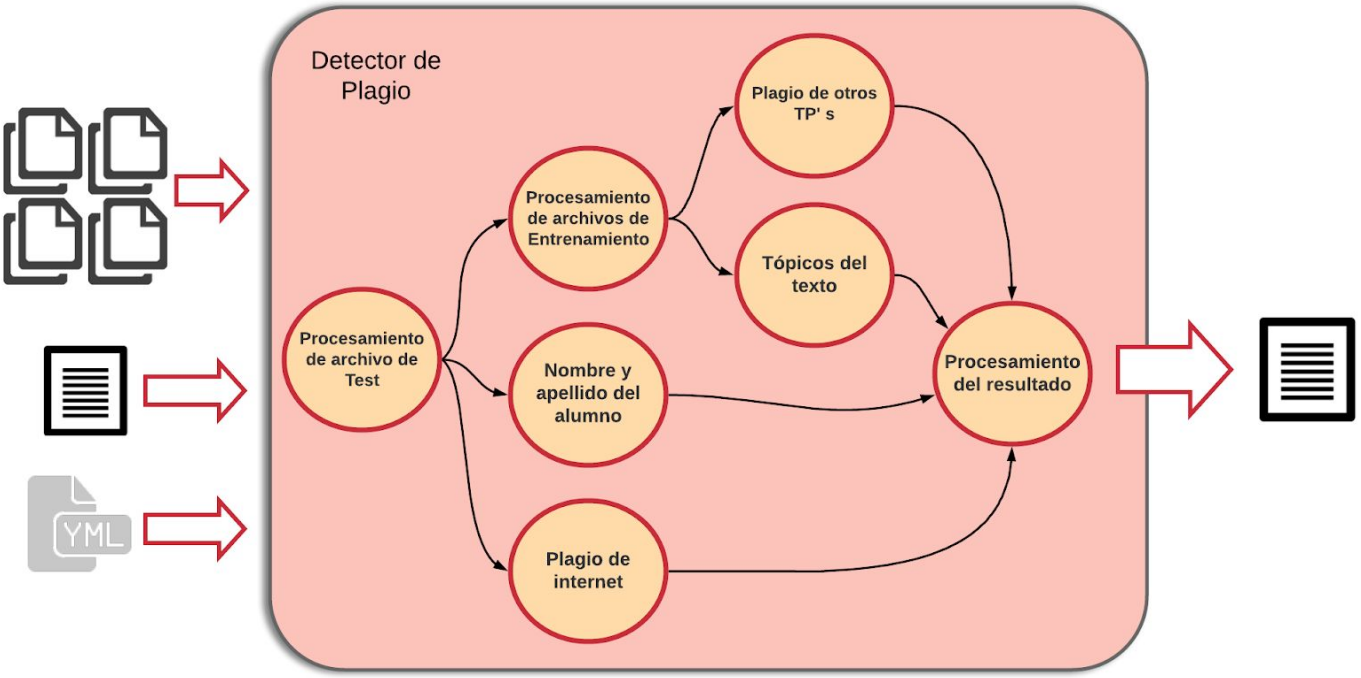
Comencé el trabajo práctico utilizando Jupyter Notebook, ya que es una herramienta de trabajo cómoda para proyectos de NLP, porque nos brinda resultados parciales a medida que vamos desarrollando nuestro código.

Llegado un momento me encontré con la necesidad de realizar funciones, loguear resultados y que posteriormente mi resolución pueda ser fácilmente reproducible en cualquier entorno, siendo a veces el usuario final una persona que le interesa más la entrada y la salida del detector de plagio y no toda la resolución, por lo que decidí migrar al entorno de desarrollo que nos ofrece Jetbrains para Python, PyCharm.

El código fue escrito en Python 3 y utilicé varias librerías que iré detallando en cada componente. Para que las librerías que instalé solo afecten a este proyecto y no a algún otro a futuro, cree un entorno con MiniConda como recomendó la cátedra.

El código se encuentra en *~repositorio/src/python* separado en distintos archivos y en distintas funciones según cada componente del trabajo práctico.

COMPONENTES GENERALES



ENTRADA Y SALIDA DEL ALGORITMO, Y CÓMO EJECUTARLO

Para ejecutar el programa debemos inicialmente prestarle atención a ciertas configuraciones que se encuentran en el archivo config.yml (¿Por qué YML? Porque es un formato de archivo de configuración cómodo para trabajar con Python)

El archivo cuenta con los siguientes campos:

- path_archivos_entrenamiento: ubicación de los archivos o trabajos prácticos anteriores (dataset) donde se desee consultar el plagio
- path_archivo_test: ubicación del archivo del que querremos detectar el plagio
- path_resultado: ubicación donde querremos almacenar el archivo resultado del algoritmo de detección de plagio
- cantidad_de_topicos: cantidad de palabras clave, tema, o tópicos del texto de test que querremos que el algoritmo nos diga para luego poder identificar el tema más fácilmente
- cantidad_de_links: cantidad de links de google donde querremos que el detector busque si hay plagio (por oración)
- buscar_en_pdfs: True o False. Si queremos que, en caso de que uno de los links donde se busca plagio sea un pdf que haya que descargar, se busque plagio ahí también o no

Luego de setear el archivo de configuración deberemos poner en la ubicación que seteamos en el path_archivos_entrenamiento todos los trabajos prácticos anteriores y documentos en alguno de los siguientes formatos: .txt, .doc, .docx, .pdf, .ppt, .pptx

Los documentos que no sean de alguno de esos formatos no se tendrán en cuenta.

Luego deberemos poner el archivo a analizar en la ubicación que seteamos en el path_archivo_test (debe ser de alguno de los formatos anteriores).

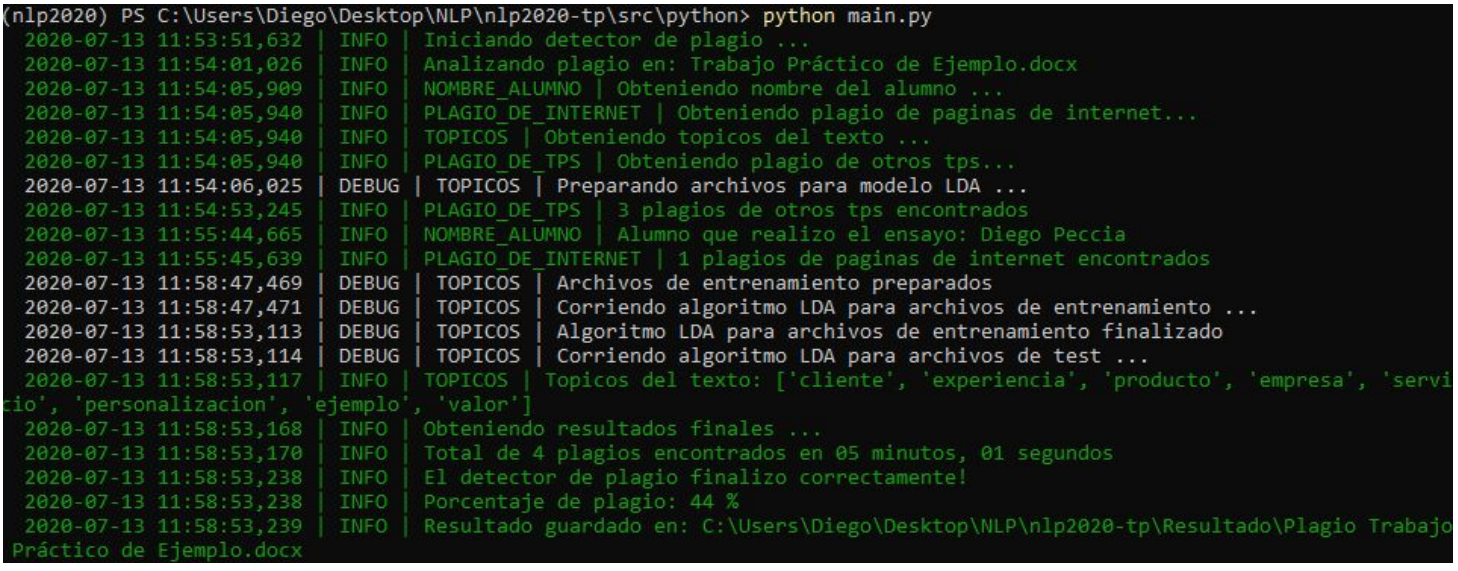
Opcionalmente podemos agregar en el archivo *"Texto excluido de plagio.txt"* texto que no nos gustaría que se analizara el plagio. Está más que nada pensado para consignas de trabajos prácticos. En caso de que no deseemos excluir nada, dejamos el archivo vacío.

IMPORTANTE: Si es la primera vez que se va a ejecutar el algoritmo previamente deberemos ejecutar el archivo librerias.bat ubicado en la carpeta del repositorio, que nos instalará todas las librerías que necesitamos para que el detector de plagio funcione correctamente. Si se creo un ambiente en miniconda, este bat se debe ejecutar en el Anaconda Powershell Prompt, con el ambiente previamente activado.

Finalmente para ejecutar el algoritmo, ponemos en el Anaconda PowerShell Prompt, con el ambiente previamente activado, posicionados en el directorio ~repositorio/src/python/ :

python main.py

Podremos ir viendo las etapas por las que va pasando el algoritmo en los logs de la consola de la siguiente forma:



El resultado se guardará en un archivo en la ubicación que seteamos en el path_resultado con el nombre de *Plagio <nombre_del_tp>.pdf* en el que se podrá ver:

- Nombre del archivo de texto procesado

- Tópico o Tema del texto procesado
- Nombre y apellido del alumno que realizó el TP
- Cantidad de Plagios encontrados y cuanto tiempo se tardó
- Porcentaje de plagio del texto en general
- Listado de frases (y su ubicación dentro del texto) que podrían ser plagios de otros TPs previamente subidos o bien copiados de internet en una tabla con las siguientes columnas:
    - ➢ Oración plagiada (oración del archivo de texto procesado)
    - ➢ Oración original (oración del lugar donde se plagió)
    - ➢ Lugar donde se encontró (con un hipervínculo al archivo o página de internet donde se encontró)
    - ➢ Ubicación dentro del archivo en el que se encontró

Ejemplo de resultado del plagio en el repositorio, carpeta /Resultado/Plagio Trabajo Práctico de Ejemplo.pdf

DESCRIPCIÓN DE CADA COMPONENTE

En este apartado se explicará detalladamente cómo funciona cada componente, que responsabilidades tienen, problemas que fueron apareciendo y librerías usadas.

Procesamiento de archivos:

El algoritmo comienza obteniendo todos los archivos de entrenamiento y el archivo de test y procesandolos para su posterior manejo.

¿Qué implica procesarlos?

Se levantan todos los archivos de su correspondiente directorio y, los tipos de archivos soportados se parsean a texto y se guardan en una clase ArchivoTxt, que funciona como una Struct de C, es decir, solamente es para almacenar cómodamente los datos para ser utilizados, en este caso un Struct de Nombre (nombre del archivo), Extensión, y Texto. Los tipos de archivos no soportados se descartan.

Luego de levantarlos se limpian todos los archivos y se obtiene de cada uno de ellos un array de oraciones, que será mucho mejor para que los procesen los siguientes componentes. Dividir el texto en oraciones lo hago con la función sent_tokenize de nltk al que se le puede especificar el lenguaje en el que está el texto para que se tengan en cuenta además de puntos, signos de pregunta, exclamación y demás a la hora de dividir.

La limpieza de los archivos la separé en hilos (uno por archivo) para que se realice más rápido. Esta limpieza fue hecha con expresiones regulares y implica:

- Convertir múltiples enters (\n+) en un enter solo

- Convertir un enter (\n) en un punto (para que sent_tokenize tome a los enter como puntos)
- Convertir múltiples espacios a un espacio solo
- Letras con tilde a su correspondiente letra sin tilde
- Normalizar todo a unicode para que no queden caracteres extraños que puedan haber aparecido al momento del parseo, como emoticones, o imágenes. Esto se hizo con la librería unicodedata2

Y más cosas que dependen del lenguaje que se esté utilizando, en este caso para español basta.

Una vez que realizamos la limpieza de los archivos, obtenemos del archivo *"Texto excluido de plagio.txt"* que tiene como dijimos antes, texto que no nos gustaría que se analizara el plagio. Está más que nada pensado para consignas de trabajos prácticos. Por lo que del archivo de test, nos quedamos con todas las oraciones que no hayan sido excluidas en ese archivo. Si el archivo está vacío, no se excluye nada.

Con este paso ya tenemos todos los archivos listos para realizar la detección de plagio, obtener el nombre del alumno, etc.

Nombre y Apellido del alumno:

En este paso tenemos el texto a procesar para analizar el plagio dividido en oraciones. Lo que queremos lograr es obtener el nombre y apellido del alumno que realizó este trabajo práctico.

Iniciamos este proceso cargando de la librería spacy el modelo en español de es_core_news_sm, que es un modelo pre entrenado de noticias de la CNN en la que se detectectaron/taggearon entidades, en este caso nos interesan las llamadas Named Entity Labels (https://spacy.io/models/es), como ser PER, LOC, ORG, entre otras. Traducido, este modelo ya tiene posibles nombres de personas taggeados o nombres propios, que es lo que nos interesa.

Una vez que tenemos este modelo precargado se lo aplicamos a cada oración de nuestro archivo de test y filtramos los que sean PER (entidades reconocidas por el modelo como posibles personas/nombres propios.

Luego de esta lista de entidades_reconocidas_como_personas, las pasamos a minúsculas y nos quedamos solamente las que no sean Stopwords. Las stopwords son palabras frecuentemente usadas para acompañar a otras más importantes, son las palabras que en una oración no van a ser relevantes. Esto lo hacemos porque el modelo de Spacy no es perfecto, y puede haber detectado cosas que estaban mal. Las stopwords las obtenemos también con NLTK.

Luego de cada una de estas entidades, que posiblemente no sea una lista muy grande, hacemos algo muy interesante, que es obtener el contexto en el que se encuentra dicha

entidad en el texto. Ejemplo, si la entidad reconocida es Will Smith, su contexto se mostraría de la siguiente forma:

… En 1990, la fama de Will Smith aumentó drásticamente, cuando protagonizó la serie de televisión El Principe de Bel-Air …

De esta manera una vez que tenemos el contexto para cada entidad, filtramos las entidades en cuyos contextos (a la izquierda de la entidad) figure alguna palabra como: alumno, alumna, integrante, apellido, nombre y devolvemos la primera entidad que matchee con estas condiciones, ya que posiblemente el nombre del alumno esté en la primera página del tp.

En caso de no encontrar el nombre del alumno pero si entidades que podrían serlo, loggeamos las entidades que encontramos, y si no encontramos nada, loggeamos que no se pudo encontrar el nombre del alumno.

Problemas con la obtención del nombre del alumno:

Claramente este componente no es perfecto, sencillamente porque cada humano tiene una manera distinta de escribir su nombre en un trabajo práctico. Si bien la mayoría pone Alumno: o Nombre del alumno: o Integrante: o pone su apellido en el título del trabajo práctico (aunque el título podría ser “TP Will Smith Diego Peccia.pdf”, ahí como se si Will Smith es el nombre del alumno o Diego Peccia? Es complejo), conozco un caso de un compañero de la facultad que pone su nombre en el pie de página de sus tps, solamente su nombre y apellido sin nada más. Para cualquier algoritmo sería complicado detectar este caso, ya que podría tratarse del nombre de un profesor, de alguien a quien se haga referencia dentro del texto, un autor, etc. Ni hablar de la cantidad de nombres distintos que existen hoy en día, como el hijo de Elon Musk que se va a llamar XÆA-12 !! ¿Cómo hacemos para detectar ese nombre con nuestro algoritmo cuando éste haga un trabajo práctico?

Otro tema es que el pos_tagging de spacy en español no es tan preciso como lo es en inglés.

Lo que quiero decir con esto es que si bien este algoritmo detecta una gran cantidad de casos, no es perfecto.

Tópicos del texto:

Para este componente esperamos a que terminen todos los hilos de procesamiento de archivos para poder arrancarlo, ya que los necesitaremos.

Para obtener el tema o tópicos del texto utilicé Topic Modelling, que se basa en obtener un modelo estadístico de los tópicos de un texto, también llamadas palabras clave, temas principales, etc.

En particular mirando los resultados obtenidos en este post de Medium: https://medium.com/@armandj.olivares/topic-modeling-on-spanish-text-f7a5e998fb90 decidí usar el modelo LDA para realizar el Topic Modelling sobre el texto a analizar.

¿Qué es el modelo LDA y en que se basa?

El modelo de Asignación Latente de Dirichlet (LDA) es un modelo de obtención de tópicos el cual se entrena con ciertos archivos. El modelo ve a cada archivo como una serie de categorías o tópicos distribuidos en base a la distribución de Dirichlet (familia de distribuciones de probabilidad continuas multivariada según Wikipedia). Este modelo está fuertemente relacionado con Bag of Words, ya que no le importan el orden en el que estén las palabras sino la cantidad de las mismas en el texto.

¿Qué realizo entonces para obtener los tópicos del texto?

Básicamente lo primero que hago es preparar cada archivo de entrenamiento para entrenar el modelo LDA. A cada archivo lo preparo en un hilo aparte para tener paralelismo de tareas. “Prepararlo para LDA” consiste en quedarme solamente con los sustantivos del texto y lemmatizados, es decir, con su palabra raíz (si la palabra es jirafas, lemmatizada sería jirafa). Esto lo hago nuevamente con spacy, con el modelo de es_core_new_sm.

Luego con la librería Gensim, creo un diccionario que contiene la cantidad de veces que aparece una palabra en los archivos de entrenamiento y con Gensim doc2bow creo una lista que contiene la cantidad de palabras de cada archivo y cuantas veces aparecen.

Ahora ya tengo listo todo lo que necesito para entrenar el modelo LDA, por lo que lo entreno.

Luego de entrenarlo le paso un archivo que desconoce el modelo (el archivo del cual queremos saber sus tópicos) y lo aplico al modelo entrenado. Esto me va a devolver varias listas de posibles tópicos junto con un puntaje asociado, por lo que me quedo con la lista de tópicos que más puntaje tenga, y me quedo con los primeros n tópicos, siendo n la cantidad de tópicos que definimos en el archivo de configuración.

Problemas con la obtención del tópico del texto:

La lemmatización esta ya bastante bien lograda para el idioma inglés, pero no es tan perfecta en el idioma español, por lo que algunas palabras podrían no quedar tan bien lemmatizadas.

Otra cosa es que inicialmente lo que quise hacer es obtener el Tema del texto, no los tópicos, algo así como “Marketing en Internet” no obtener “marketing, internet, economía, ...”. Esto está bastante hecho para el idioma inglés, sin embargo en español no. Para poder lograr esto, lo que se necesitaría es un dataset muy grande que dado un conjunto de palabras claves o tópicos les corresponda un Tema como “Naturaleza” o “Macroeconomía”.

Plagio de otros TP's:

Tanto para este componente como el de Plagio de Internet, para la parte de comparar una oración con otra para saber si es plagio o no, utilicé el método de Similitud Del Coseno, aunque se podrían haber utilizado otros como Jaccard, o Levenshtein. Leyendo en Medium y TowarsDataScience diferentes resultados con los distintos métodos concluí que con la Similitud del Coseno se obtienen resultados suficientemente favorables para lo que queremos lograr.

"La similitud del coseno es una medida de la similitud existente entre dos vectores en un espacio que posee un producto interior con el que se evalúa el valor del coseno del ángulo comprendido entre ellos" (Wikipedia - Similitud del Coseno)

Para obtener el plagio de otros trabajos prácticos, lo que hacemos es dado nuestro archivo del cual queremos detectar plagios ya previamente separado en oraciones, preparamos cada oración para ser comparada y la comparamos con cada oración (preparada para comparar) de cada archivo de entrenamiento (los "otros tp´s"). Esto es mucho procesamiento pero es muy preciso, por lo que dividimos este trabajo en hilos, un hilo por cada oración del archivo de test.

De cada oración voy obteniendo el porcentaje de similitud según el algoritmo mencionado anteriormente y voy guardando en una lista como (oracion, oracion_mas_parecida, mayor_porcentaje, archivo_donde_se_encontro, ubicacion_dentro_del_texto), siendo la ubicacion_dentro_del_texto un rango de palabras entre las que se encuentra la oración:

Por ejemplo, si el texto del que se encontró plagio tiene 1000 palabras, una oración podría estar en el rango de las (500,509) palabras de todo el texto.

Luego filtro los elementos de la lista cuyo porcentaje de plagio sea mayor a 0.7, que probando con varios porcentajes concluí que con 0.7 se agarran la mayor cantidad de plagios sin que sean cosas que no serían consideradas plagio.

Anteriormente mencioné lo de "preparar las oraciones para ser comparadas". Esto es básicamente sacar las stopwords, cambiar letras con tilde por su correspondiente sin tilde, pasar todo a minúsculas, quedarme solo con los caracteres alfanuméricos y no analizar plagio si la oración sin stopwords:

- Es un titulo: un titulo no sería considerado plagio (Títulos son los que todas sus palabras comienzan con Mayúscula y siguen con minúscula)
- Si tiene 3 o menos palabras (demasiado pocas para poder considerarla como plagiada)
- Si es una consigna o enunciado de un trabajo práctico (no debería ser considerada plagio)

De esta manera obtenemos resultados de plagio de otros tps más precisos.

Plagio de Internet:

Lo que realiza este componente es bastante similar al anterior componente, salvo que no depende de que el componente del procesamiento de los archivos de entrenamiento haya terminado ya que no los necesita, por lo que comenzará a ejecutarse una vez que el archivo de test esté procesado.

Básicamente lo que hacemos es dado nuestro archivo del cual queremos detectar plagios ya previamente separado en oraciones, preparamos cada oración para ser comparada y la buscamos en Google con la API de googlesearch. La oración ya está preparada para ser comparada como dijimos en el componente anterior.

Luego, de cada oración, nos quedamos con los primeros n links que resultaron de buscarla en Google, siendo n el número que definimos en el archivo de configuración para el campo cantidad_de_links. Guardamos el html de cada una de esas páginas y las convertimos a texto con la herramienta de BeautifulSoup.

Luego, comparamos la oración con cada oración (preparada para comparar) de cada una de las páginas. Esto nuevamente es mucho procesamiento, por lo que dividimos este trabajo en hilos, un hilo por cada oración del archivo de test.

De cada oración voy obteniendo el porcentaje de similitud según el algoritmo mencionado anteriormente y voy guardando en una lista como (oracion, oracion_mas_parecida, mayor_porcentaje, url_donde_se_encontro, ubicacion_dentro_de la pagina).

Luego filtro los elementos de la lista cuyo porcentaje de plagio sea mayor a 0.7, tal como hicimos con el componente anterior.

Adicionalmente, dejé la posibilidad de setear en el archivo de configuración en el campo buscar_en_pdfs (Booleano) si se quiere que si una página en realidad es un link a un pdf, busque el plagio en el pdf o no. ¿Por qué hice esto? Sencillo, porque está la posibilidad de que el pdf sea un Paper Académico o un libro de 300 páginas, el cual tardaría mucho en descargarse, por lo que el detector de plagio podría estar mucho tiempo ejecutándose. Por otro lado, obviamente esto es más preciso, al igual que si buscamos en 5 paginas de internet por oración, o en 20, o en 1000, sería muy preciso, pero el tiempo nos jugaría en contra. Por ello dejé a decisión del usuario la cantidad de páginas en las que quiere buscar y si desea buscar en pdfs o no.

En este punto lo que se podría haber hecho también es que por cada oración por cada página, el algoritmo haga el equivalente al CTRL+F (Find) de la computadora para que sea más rápida de buscar la oración en la página. Opté por no dejar de lado esa opción y comparar cada oración preparada para comparar con el método de similitud del coseno ya que es mucho más preciso y se obtienen resultados mucho más favorables.

Problema interesante que surgió en plagio de internet:

Algo muy interesante que surgió de utilizar la API de GoogleSearch es que la función search, cuenta con un parámetro llamado pause:

search(oracion_preparada, tld="com.ar", num=config[“cantidad_de_links”], stop=config[“cantidad_de_links”], ***pause****=2)*

Este parámetro nos indica la cantidad de milisegundos a esperar entre cada Request. Si se pone un número muy alto, esto será lento, pero si se pone un número muy bajo y se hacen muchos pedidos, nos lanzará una excepción con el código “Too Many Requests”. Con 2 milisegundos alcanza para lograr velocidad y eficacia.

El problema vino cuando pusimos un hilo por oración: Cada hilo realiza x pedidos a la API que podrían llegar a ser simultáneos mientras más oraciones hayan.

Esto lo solucioné poniendo un blockeo de tipo Mutex global sobre esta función search con un sleep de 2 milisegundos. De esta manera, cada vez que un hilo quiera hacer un llamado, primero verificará si el Mutex esta bloqueado, esperará 2 milisegundos a que se desbloquee y luego hará el llamado. De esta manera forzamos a que este recurso se utilice cada 2 milisegundos al menos y no nos lance la excepción de Too Many Requests.

Parafraseo y otros problemas en la detección de plagio:

¿Qué tan preciso es el algoritmo en cuanto a parafraseos? ¿Hasta qué punto se considera que una oración sigue siendo un parafraseo de otra o que sencillamente no había otra forma de escribirla o explicarla?

Estas son preguntas complicadas a la hora de realizar un detector de plagio. Se nos pidió que el algoritmo tenga la posibilidad de detectar parafraseos. Investigando descubrí que una oración puede tener miles de parafraseos distintos (formas de escribirla) y que no está del todo claro hasta qué punto la oración cambió tanto que ya no se considera parafraseo.

La mejor manera de lograr esto que estuve analizando es tener un banco de sinónimos para cada palabra importante de una oración, obteniendo por ejemplo el banco de sinónimos desde Wordnet, y luego realizar una combinatoria de todas las posibles oraciones parafraseadas, es decir, todas las combinaciones de sinónimos de cada palabra importante del texto, con lo que se conoce como Text Spinner, que, si bien en Inglés hay más opciones en cuanto a Text Spinners, en español los que hay son pagos, y no son perfectos, ya que nuestro idioma es bastante complejo.

Es por esto que si tengo un trabajo práctico de 100 oraciones y de cada oración obtengo unas 10 parafraseadas, la cantidad de oraciones a procesar se me va a 1000, de las cuales cada una compararla con la similitud del coseno o cualquier otra con las oraciones de los otros tps, y cada una buscarla en internet y compararla con cada oración de n páginas de internet se hace inviable para el procesamiento con el que contamos, más teniendo en

cuenta que uno quiere tener un resultado del Detector de Plagio lo antes posible, más si quisiéramos venderlo como producto.

Por todo esto que cuento es que me parece algo inviable que el algoritmo pueda detectar parafraseos, más que nada teniendo en cuenta que hoy son parafraseos, pero el día de mañana podríamos querer que el detector de plagio no solo busque la oración en Google con resultados en español, sino traducir cada oración a idiomas muy usados como el inglés, alemán o chino y realizar la búsqueda en esos idiomas en n páginas de internet más, para saber si el alumno plagio cosas de otro idioma, lo que nuevamente, sería una cantidad de procesamiento abismal.

Procesamiento del Resultado:

Finalmente, luego de que los hilos de nombre y apellido del alumno, tópicos, plagio de otros tps, plagios de internet finalicen, realizamos un informe con la librería docx en el que guardamos todos los resultados obtenidos. Este informe se guarda en la carpeta que definimos en el archivo de configuración en el *path_resultado*. Como esta compuesto este informe ya fue detallado en la parte de Entrada y Salida del Algoritmo, en este mismo documento.

En el caso de que para una oración se encuentre más de un plagio, siempre decidimos quedarnos con el primero, y si se encontró plagio en otro tp y también de internet decidimos poner en el informe solamente el de otro tp, ya que si el plagio es de otro tp nada nos indica que a su vez la otra persona no haya plagiado de internet.

En esta parte también filtramos de la lista de plagios los que hayan sido correctamente citados y no los consideramos plagio.

Algo interesante a destacar es que incluimos un hipervinculo a los archivos o páginas de los cuáles hubo plagio, para que sea más fácil de revisar éste mismo.

Una breve aclaración sobre la PERFORMANCE

Este tema ya lo fui comentando en los puntos anteriores, lo único que quería aclarar era que en un principio se realizaba todo en un mismo hilo, tardando hasta 24 minutos en detectar el plagio de un tp. Una vez que decidí agregar hilos y un timeout para las páginas de internet que tarden mucho en responder, este tiempo para el mismo tp pasó a ser de 6 minutos.

CONCLUSIONES Y TRABAJOS FUTUROS

Como conclusiones puedo decir que este proyecto siempre va a poder mejorarse, y nunca va a ser perfecto, así como nosotros no somos perfectos tampoco.

Como dije antes, el idioma español es un idioma complejo, del cual siempre aparecen cosas nuevas, y ¿cómo vamos a hacer para que un algoritmo sepa todo lo que pasa en un lenguaje como el nuestro si ni nosotros lo sabemos? Me refiero a que constantemente la gente va cambiando el lenguaje a su gusto según su cultura, preferencias, molestias, etc.

El campo del procesamiento del lenguaje natural se está expandiendo cada vez más y siempre busca ser lo más cercano al cerebro humano posible, pero por todas las cosas que fui comentando nos demuestra que sigue estando lejos.

Como proyecto me pareció muy interesante, la verdad es que le dediqué bastante tiempo y aprendí muchas cosas, entre ellas Python, NLTK, el concepto de entrenamiento y testeo, y modelos como el LDA y la similitud del coseno.

Trabajos futuros pueden ser tranquilamente diseñar una interfaz gráfica de usuario ya sea web, mobile, o desktop, que por atrás corra este algoritmo, para que sea usada en colegios y facultades, así como también se podría hacer que corra en la Nube y comprar muchos recursos para que podamos explotar más la parte del parafraseo, la traducción y la cantidad de páginas de internet sin preocuparnos tanto por la performance.

BIBLIOGRAFÍA

- https://www.nltk.org/book/
- https://medium.com/@yeralway1/primeros-pasos-en-nlp-con-spacy-un-vistazo-general-734686843a57
- https://www.it-swarm.dev/es/python/calcular-la-similitud-de-las-dadas-cadenas-de-2-oraciones/1070875781/
- https://es.wikipedia.org/wiki/Similitud_coseno
- https://www.geeksforgeeks.org/performing-google-search-using-python-code/
- https://www.pluralsight.com/guides/extracting-data-html-beautifulsoup
- https://stackoverflow.com/questions/384076/how-can-i-color-python-logging-output
- https://stackoverflow.com/questions/775049/how-do-i-convert-seconds-to-hours-minutes-and-seconds
- https://medium.com/@armandj.olivares/topic-modeling-on-spanish-text-f7a5e998fb90
- https://stackoverflow.com/questions/60534999/how-to-solve-spanish-lemmatization-problems-with-spacy
- https://medium.com/@armandj.olivares/topic-modeling-on-spanish-text-f7a5e998fb90
- https://towardsdatascience.com/topic-modeling-and-latent-dirichlet-allocation-in-python-9bf156893c24
- https://ri.itba.edu.ar/bitstream/handle/123456789/1250/TFI_Hammoe.pdf?sequence=1&isAllowed=y
- https://python-docx.readthedocs.io/en/latest/user/styles-understanding.html
- https://stackoverflow.com/questions/47666642/adding-an-hyperlink-in-msword-by-using-python-docx
- https://stackoverflow.com/questions/39418620/extracting-text-from-multiple-powerpoint-files-using-python/51905465