Detección de plagio para trabajos universitarios

Autor: Marcos Infantino

Abstract

El plagio es el uso del trabajo o ideas de otras personas como si fueran de uno mismo, apropiándose el crédito de la creación (Chen et al., 2010).

La detección de plagio en los trabajos académicos es un problema que se hace cada vez más grande, a causa de la gran y creciente cantidad de información que existe en Internet, a la facilidad de encontrarla utilizando navegadores y a la entrega electrónica de los trabajos (Rico Juan et al., 2016).

En el presente trabajo se propondrá un algoritmo de detección de plagios de trabajos prácticos de alumnos en base a un dataset, el cual contendrá trabajos de años anteriores.

Palabras clave

Detección, plagio, algoritmo, documentos

1. Introducción

Actualmente, existen diferentes tipos de plagio: plagiar textos solo traduciéndolos, reutilizar ideas sin citar, o incluso comprar trabajos con certificados de garantía de no ser detectados por programas anti plagio (Tripathi, 2009). Esto representa un grave problema para el ambiente académico, ya que se dificulta cada vez más la detección de copias.

"La mayoría de los tipos de plagio son difíciles de detectar incluso para expertos debido a su sofisticación para enmascarar las ideas o párrafos originales de los que provienen."(Rico Juan et al., 2016).

Ante este problema, es necesario un desarrollo constante y sostenido de aplicaciones de detección de plagio cada vez más sofisticadas. El uso de herramientas de detección de plagio proveen un fuerte apoyo al ámbito académico, y es por este motivo que en países desarrollados como, por ejemplo, Estados Unidos, su uso se encuentra ampliamente extendido y aceptado en la comunidad académica (Rico Juan et al., 2016).

Existen dos tipos principales de análisis de plagio: el intrínseco y el extrínseco. El análisis de plagio extrínseco compara un documento candidato con un corpus de documentos referencia, y trata de encontrar similitudes con respecto a los mismos. Por otro lado, el análisis de plagio intrínseco no utiliza ninguna colección de referencia, y trata de determinar si existe plagio a través del análisis en los cambios de estilo dentro del documento (Seaward & Matwin, 2009).

El enfoque de este trabajo es el desarrollo de un algoritmo de detección de plagios que utiliza un análisis extrínseco, utilizando el lenguaje Python y las bibliotecas que provee la comunidad. Además, el algoritmo podrá reconocer plagios literales extraídos de la web. De esta manera, se prosigue a estructurar el artículo de la siguiente manera: en la sección 2 se hablará sobre la base de documentos utilizada para el entrenamiento del algoritmo; en la sección 3 se hablará sobre el entrenamiento del algoritmo para el reconocimiento de plagio en un nuevo documento; en la sección 4 se discutirá sobre las técnicas utilizadas para la detección de plagio con respecto al dataset; en la sección 5 se expondrá el algoritmo de búsqueda de plagios literales en la web; en la sección 6 se expondrán los testeos realizados al algoritmo para verificar su correcto funcionamiento; y en la sección 7 se proseguirá con las conclusiones y futuras líneas de trabajo.

2. Dataset de documentos

El dataset utilizado consiste en una serie de trabajos realizados por alumnos para una materia universitaria. Los mismos fueron compilados a través de años y las sucesivas entregas de los estudiantes sobre diferentes temas. De esta manera, este dataset nos permitirá detectar plagio entre los mismos

trabajos del dataset, o incluso entre nuevos trabajos de la misma materia con respecto a los de años anteriores.

3. Entrenamiento del algoritmo

Los documentos del dataset se encuentran en distintos formatos, entre los cuales se destacan .doc, .docx, .ppt, .pptx y .pdf. Los mismos son transformados a texto plano a través de las bibliotecas python–docx, python–pptx, pdfplumber y tika.

Una vez que se obtiene el texto plano de un documento, se lo pasa a un string, y el mismo pasa a ser dividido en párrafos.

Cada párrafo de cada documento será pre procesado, lo que significa que pasará por una tokenización, remoción de signos de puntuación, conversión a minúscula, clasificación POS, remoción de stopwords y stemming. De esta manera, cada párrafo contendrá un metadata, el cual es la lista de tokens resultantes del preprocesamiento sobre el texto real.

El preprocesamiento mencionado me brindará las siguientes ventajas:

- La remoción de stopwords y signos de puntuación me permitirán evadir falsos positivos, ya que ni las stopwords ni los signos de puntuación sumarán en el puntaje final de similitud entre dos documentos.
- El stemming me permitirá llevar todas las palabras hacia su raíz común, lo que me permitirá compararlas con mayor facilidad. Por ejemplo: “camina” y “caminó” son convertidas a “camin” por el Snowball Stemmer de nltk, que es la biblioteca que se utilizará para tal fin.
- La clasificación POS me permitirá verificar que dos palabras iguales hayan sido usadas con el mismo sentido dentro de la oración, evitando así falsos positivos.
- La conversión a minúscula es necesaria, ya que evita falsos negativos en la comparación entre dos palabras cuya única diferencia es el uso de letras mayúscula. Por ejemplo, “Enero” y “enero”.

Las herramientas utilizadas para el procesamiento fueron las siguientes:

- Biblioteca ***nltk***: la misma se utilizó para la tokenización y reconocimiento de stopwords y signos de puntuación (Bird et al., 2009).
- Biblioteca ***spacy***: la misma se utilizó para la clasificación POS de cada uno de los tokens del texto (Explosion AI, 2016).

De esta forma, se genera un objeto Doc por cada uno de los documentos del dataset. Dicho objeto contendrá el texto plano, así como una lista de objetos Parragraph. Un objeto Parragraph contendrá el texto plano vinculado al mismo y, además, el metadata vinculado a ese texto (recordar que se le llama metadata al texto ya procesado).

En las imágenes 1 y 2 se pueden observar la conformación de los objetos Doc y Parragraph respectivamente:

```
class Doc:
    name = ""
    text = ""
    fileType = ""
    author = ""
    paragraphs = []
```

Imagen 1

```
class Parragraph:
    text = ""
    sentences = []
    metadata = []
```

Imagen 2

Realizar dicho preprocesamiento sobre cada uno de los documentos del dataset conlleva un tiempo extenso. Por este motivo, se

decidió cachear todo el dataset pre procesado en pequeños archivos, para que así el algoritmo pueda valerse directamente de estos, y no tener que pre procesar todo el dataset cada vez que va a analizar un nuevo documento.

Para realizar esto, se utilizó la biblioteca ***pickle*** (Anónimo, 2018), la cual permite crear un dump de un objeto dentro de un archivo. Como se explicó, cada documento del dataset se corresponde con un objeto Doc. De esta manera, se crea un dump de todos los objetos Doc generados luego del preprocesamiento, obteniendo como resultado un pequeño archivo caché por cada documento del dataset.

4. El algoritmo de detección de plagio extrínseco

El primer paso a realizar será la obtención de un objeto Doc a partir del documento candidato. Luego de esto, se cargan los documentos del dataset a partir del caché. A continuación, se itera sobre cada uno de los documentos del dataset (documentos referencia), para así compararlos con el documento candidato.

De esta forma, el documento candidato se comparará con cada uno de los documentos referencia. Cada párrafo del documento candidato será comparado con todos los párrafos del documento referencia. Esta comparación se realizará mediante una versión modificada del Jaccard Similarity Score, en la cual también se tiene en cuenta el POS tagging (Adam & Suharjito, 2014).

El Jaccard Similarity Score se define, tal como se muestra en la imagen 3, como el cociente entre la cardinalidad de la unión de dos conjuntos y la cardinalidad de la intersección de esos conjuntos(Chen et al., 2010).

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}$$

Imagen 3

Para el caso de este trabajo, los conjuntos a comparar a través del Jaccard Similarity Score serán los metadata de los párrafos de los documentos. Se definirá, de esta manera, como la intersección de una metadata A y una metadata B a aquellos tokens que se encuentren en ambos conjuntos y a la vez posee el mismo POS tag. Esto significa, que las palabras no solo serán comparadas por su similitud textual, sino que también se tendrá en cuenta su POS, buscando asegurar que se hayan usado de la misma forma en los dos párrafos.

Es importante aclarar que aquellos párrafos que se encuentren citados poseerán un Jaccard Similarity Score de 0. Por otro lado, cabe destacar que también existe la posibilidad de excluir a las consignas y preguntas de la comparación del documento a través del archivo de configuración.

El proceso se repite con cada uno de los documentos del dataset. Una vez que se terminó la comparación de todos los párrafos del documento candidato con los párrafos del dataset, se le asignara a cada párrafo el máximo Jaccard Similarity Score que haya obtenido en una comparación, para luego realizar un promedio por párrafos en el documento y obtener el nivel de plagio general del documento candidato con respecto al dataset.

Una vez terminado el análisis, la aplicación mostrará por pantalla cada uno de los párrafos del documento candidato que son sospechosos de plagio, junto con un porcentaje de plagio y el párrafo referencia del cual puede ser una posible copia.

5. Algoritmo de detección de plagio en la web

Existen textos en los que el plagio no consiste de un parafraseo, sino que es una copia directamente textual al original. Para estos casos, podemos valernos de las posibilidades que nos ofrecen buscadores como Google, Bing o Yahoo, que nos permiten buscar la ocurrencia de frases literales en páginas de la web.

La estrategia de este algoritmo consiste en dividir el texto de un documento en grupos de caracteres para luego hacer una búsqueda literal de la cadena. En el caso del algoritmo presentado, se divide el texto del documento candidato en fragmentos de 100 caracteres. Esta cantidad de caracteres es ideal, ya que es suficiente para evitar falsos positivos en las búsquedas y, además, es soportado por la mayoría de los buscadores (Rico Juan et al., 2016).

La herramienta utilizada para la realizar la consulta a los buscadores fue la biblioteca ***Beautiful Soup.*** El algoritmo brinda la posibilidad de realizar las consultas a través de Google o a través de Bing, y esto puede ser cambiado mediante el archivo de configuración. Se utiliza la biblioteca Beautiful Soup para obtener el html de la página generada por la query del fragmento de texto buscado, y para luego procesar ese html y obtener los resultados buscados (Rafael Zambrano, 2019).

El problema que posee esta técnica es el hecho de que los buscadores detectan que se realizan muchas peticiones desde el mismo IP y, en consecuencia, restringen el número de accesos. Ante esto, se propuso una espera de 2 segundos entre cada petición, para así no generar una cantidad de peticiones muy grandes en poco tiempo. A su vez, es importante destacar que el buscador Bing es el que posee menos restricciones con respecto al límite de consultas abiertas, por lo cual se recomienda su uso (Rico Juan et al., 2016).

Una vez que se obtienen los resultados de todos los fragmentos se devuelve un índice de plagio web, (cantidad de fragmentos encontrados / cantidad total de fragmentos buscados) junto con un informe sobre qué fragmento fue encontrado y la/s referencia/s a la/s página/s donde se lo encontró.

6. Testing

Para el testeo de la aplicación se utilizó como framework de testeo ***pytest*** (CodinEric, 2020).

Primer caso de testeo: se crean dos instancias de objeto Doc a partir del documento “Trabajo Práctico 1 – Hernan Dalle Nogare.docx” y se los compara entre sí. Como resultado, se obtiene un puntaje del 100%, lo cual es correcto, ya que son dos documentos idénticos.

Segundo caso de testeo: se utilizan dos oraciones de ejemplo que son totalmente diferentes. En consecuencia, la comparación entre ambas debe dar un puntaje menor al 10%.

- Oración 1: “Rodrigo estaba paseando a su perro, cuando de repente vio un cerdo volar.”
- Oración 2: “A juan le gustan mucho las hamburguesas y las papas fritas.”

Como resultado, se obtiene un score del 0%, lo cual a simple vista se ve correcto, ya que luego del preprocesamiento no existe intersección entre los grupos de tokens de ambas oraciones.

Tercer caso de testeo: aquí se busca verificar que una cita sin referencia correspondiente penaliza el score final. Para esto, tomaremos las siguientes oraciones de ejemplo.

- s1 = “Rodrigo estaba paseando a su perro, cuando de repente vio un cerdo volar.”
- s1CopyWithCitation = “Rodrigo estaba paseando a su perro (Sentence1, 2021).”
- s1CopyWithoutCitation = “Rodrigo estaba paseando a su perro.”

Como resultado, se obtiene:

JaccardScore(s1, s1CopyWithCitation) = 0%

JaccardScore(s1, s1CopyWithoutCitation) = 42.85%

Cuarto caso de testeo: se le da al algoritmo una oración, y se le adjunta a ella diferentes formas de referenciar para chequear que estas sean reconocidas.

Ejemplo utilizado:

- **Oración:** “Rodrigo estaba paseando a su perro, cuando de repente vio un cerdo volar.”
- **Cita 1:** “Según [1], ” (formato ISO, se adiciona al principio de la oración).
- **Cita 2:** "(Despotovic-Zrakic et al., 2012)" (formato APA, se adiciona al final de la oración).
- **Cita 3:** "(Anónimo, 2010)" (formato APA, se adiciona al final de la oración).
- **Cita 4:** "(Sabbagh, 2010a)" (formato APA, se adiciona al final de la oración).
- **Cita 5:** "(Sabbagh, 2010b)" (formato APA, se adiciona al final de la oración).
- **Cita 6:** "("Barcelona to Ban Burqa", 2010)" (formato APA, se adiciona al final de la oración).

Quinto caso de testeo: en este caso se busca verificar que dos oraciones idénticas en textos distintos aumentan la similitud.

Ejemplos utilizados:

- Doc1 = “Trabajo Práctico 1 - Hernan Dalle Nogare.docx”
- Doc2 = “TP 3 The experience economy (1).docx”
- NuevaOracion = "Esta es una nueva oración para testear si aumenta la similitud entre ambos documentos."

Como resultado se obtiene que, mientras que los dos documentos originales dan un puntaje de similitud del 9.63%, los documentos con la oración agregada dan un puntaje de similitud del 10.91%.

Sexto caso de testeo: en este caso se busca verificar que una copia parcial de una oración penaliza menos que una copia total.

Ejemplos utilizados:

- s1 = "Spiderman salvó a la mujer que estaba atrapada bajo los escombros del edificio en llamas."
- s1CopiaParcial = "Ironman salvó al hombre que estaba atrapado dentro del edificio en llamas."

Como resultado, se obtiene:

JaccardScore(s1, s1CopiaParcial) = 36.36%

JaccardScore(s1, s1) = 100%

Séptimo caso de testeo: aquí se creará un caso de plagio artificial, modificando algunos detalles poco importantes, y se buscará tener un porcentaje de plagio mayor al 80%.

- Doc1 = “TP 1 - Larga Cola - Campassi Rodrigo .docx”
- Doc2 = “TP 1 - Larga Cola - Campassi Rodrigo(plagio) .docx”

Luego de borrar varios párrafos y agregar otros totalmente diferentes, se obtiene un puntaje de similitud del 82.7%.

7. Conclusiones

Las aplicaciones de detección de plagio son una herramienta muy útil, tanto para evitar la copia en un nivel académico, como para proteger al autor de que su propio trabajo sea robado.

El algoritmo propuesto, es un algoritmo de análisis de plagio extrínseco, que analiza el plagio en un trabajo práctico de una materia universitaria a partir de un grupo de trabajos de referencia. De esta manera, se ve con claridad cómo en el algoritmo propuesto se refleja la idea marcada de un documento

candidato que es comparado con un grupo de documentos referencia.

A partir de los casos de testeo, se puede concluir que el resultado obtenido a partir de la utilización del Jaccard Similarity Score, así como también las estrategias de preprocesamiento de tokenización, remoción de signos de puntuación, conversión a minúscula, clasificación POS, remoción de stopwords y stemming, proporcionan una fiabilidad alta para la detección de plagio entre dos documentos.

Como futuras líneas de trabajo, se propone extender el análisis de plagio de esta aplicación hacia ámbitos literarios, para así verificar su correcto funcionamiento en ese campo.

Bibliografía

Adam, A. R., & Suharjito. (2014). Plagiarism detection algorithm using natural language processing based on grammar analyzing. *Journal of Theoretical and Applied Information Technology*, *63*(1), 168–180.

Chen, C.-Y., Yeh, J.-Y., & Ke, H.-R. (2010). *Plagiarism Detection using ROUGE and WordNet*. *March 2010*. http://arxiv.org/abs/1003.4065

Rico Juan, J. R., Gallego, A.-J., & García Avilés, J. M. (2016). Estrategias para programar la detección de plagios en actividades basadas en texto. *XXII Jornadas Sobre La Enseñanza Universitaria de La Informática*, *1*, 187–194. http://rua.ua.es/dspace/handle/10045/71652

Seaward, L., & Matwin, S. (2009). Intrinsic plagiarism detection using complexity analysis. *CEUR Workshop Proceedings*, *502*, 56–61.

Tripathi, R. (2009). Plagiarism : A Plague. *Source*, 514–519.

CodinEric (2020). *Pytest: introducción a Unit Test con Python*. Youtube. https://www.youtube.com/watch?v=5ufpsjfk99U

Anónimo (2018). *Cómo usar Pickle para guardar objetos en Python*. Arregla tu pc. https://www.arreglatupc.site/index.php/tutoriales/item/5091-como-usar-pickle-para-guardar-objetos-en-python

Bird, Steven, Edward Loper and Ewan Klein (2009), *Natural Language Processing with Python*. O'Reilly Media Inc.

Explosion AI (2016). *Industrial-Strength Natural Language Processing*. Spacy. https://spacy.io/

Rafael Zambrano (2019). *Cómo hacer Web Scrapping con Python*. Open Webinars. https://openwebinars.net/blog/como-hacer-web-scraping-con-python/