Tecnológico de Monterrey

Evidencia 1. Análisis de similitud empleando Inteligencia Artificial

María Fernanda Ramírez Barragán	A01750879	ITC
Melissa Garduño Ruiz	A01748945	ITC

Fecha: 24/04/23

Instituto Tecnológico y de Estudios Superiores de Monterrey

Desarrollo de aplicaciones avanzadas

Introducción:

La detección de plagio es un proceso que busca identificar si un texto ha sido copiado o se ha utilizado información de otras fuentes sin atribuir adecuadamente la autoría. En la era digital, la facilidad de acceso a la información, la capacidad de copiar y pegar texto con facilidad e incluso parafrasear palabras o textos de un documento, han hecho que la detección de plagio sea más importante. La detección de plagio es utilizada en muchos ámbitos, desde la educación hasta la industria editorial, para asegurar la integridad y originalidad de los trabajos presentados. Con el uso de herramientas avanzadas de detección de plagio, es posible comparar un texto con una gran cantidad de fuentes en línea y detectar similitudes sospechosas que pueden indicar plagio.

Las herramientas de detección de similitud de texto son una solución eficaz para detectar posibles casos de plagio o violaciones de derechos de autor. Estas herramientas permiten comparar diferentes textos y determinar su grado de similitud a través de cálculos y determinar si uno de los textos es una copia del otro o si ha sido modificado para evitar ser detectado. Con el uso de estas herramientas, se puede asegurar la originalidad y la integridad de los trabajos académicos, científicos o literarios, así como proteger los derechos de autor de los autores y creadores de contenido.

Nuestro proyecto tiene como objetivo detectar si se presenta plagio en un conjunto de documentos brindados, comparando cada documento sospechoso con una conjunto de documentos originales empleando técnicas de aprendizaje automático, técnicas de procesamiento de lenguaje natural, técnicas de procesamiento de archivos y manejo de directorios y la técnica de similitud del coseno con la técnica de los n-gramas .

Antecedentes del Modelo Mejorado:

Para llegar al modelo final de detección de plagio, se desarrollaron varios modelos previos, uno de los cuales utilizaba la librería de Scikit-learn para realizar tareas de aprendizaje automático y convertir un conjunto de documentos en una matriz para calcular la similitud del coseno entre ellos. Este modelo consistía en enlistar todos los archivos de texto en el directorio "Texts" y almacenar sus contenidos en una lista. Luego, se definía una función para convertir una lista de cadenas de texto en una matriz de vectores. Además, se tenía otra función que tomaba dos vectores de términos normalizados y devolvía la similitud coseno entre ellos. Los resultados se almacenaban en una variable y se creaba una lista de tuplas que contenía el nombre del archivo y el vector de términos normalizado correspondiente para cada archivo.

Sin embargo, este modelo no resultó ser eficiente, ya que comparaba documentos originales con otros documentos originales, además de comparar los documentos originales con los sospechosos. Para mejorar el modelo, se crearon dos subdirectorios en el directorio "Texts" para almacenar los documentos originales y los sospechosos, y se modificó la función principal para comparar los documentos de los subdirectorios correspondientes. Aunque hubo una mejora, el modelo aún no era lo suficientemente preciso y no detectaba el plagio entre documentos con estructura similar.

Finalmente, se realizaron más modificaciones para permitir que el modelo compare un documento sospechoso con varios documentos originales y mejorar su precisión. Estas modificaciones incluyeron cambios en el cálculo de la similitud del coseno y la adición de

una función para determinar el umbral de similitud aceptable. En general, el proceso de desarrollo de este modelo de detección de plagio involucró varias iteraciones y mejoras en la metodología para lograr una mayor eficiencia y precisión en la detección de similitudes entre textos.

Modelo Mejorado:

Como se mencionó anteriormente, se tuvieron que realizar ciertos cambios en el modelo para que éste funcionase efectivamente al comparar un documento sospechoso con varios documentos originales y detectara el plagio con mayor eficiencia.

Al realizar dichos cambios, el modelo final obtenido se muestra a continuación:

La estructura de este modelo es el siguiente:

Plagarism_detection/

| -- Algorithms/

| -- __init__.py

|-- AUC.py

|-- AUC_report.md

| -- check_plagarism.py

| -- Detection_Files/

| -- Originals/

| -- Plag/

| -- Suspicious/

| -- plagarism_detection.egg-info

| -- dependency_links.txt

| --PKG-INFO

| -- SOURCES.txt

| -- top_level.txt

| -- Results_images/

| -- Unit_test/

| -- unit_tests.py

| -- __init__.py

| -- check_plagarism.py

| README.md

| -- requirements.txt

| -- Results.md

| -- run.py

| -- setup.py

Dónde:

- *“Algorithms”* Es un directorio que contiene el archivo *AUC.py,* que implementa el algoritmo de AUC (Área Bajo la Curva) para medir la precisión de la detección de plagio, y el archivo *"check_plagiarism.py*"el cual se utiliza en el archivo *AUC.py* para obtener los valores de los resultados de similitud y con ello obtener el Área Bajo la Curva. Además contiene el reporte de los resultados del Área Bajo la Curva.

- *"Detection_Files"* Es un directorio que contiene los tres subdirectorios: "Originals", "Plag" y "Suspicious". El subdirectorio"Originals contiene los archivos originales contra los que se comparan los otros dos tipos de archivos, mientras que el subdirectorio "Plag" contiene los archivos plagiados y el subdirectorio "Suspicious" contiene los archivos sospechosos y posiblemente plagiados.

- *“plagarism_detection.egg-info”* Es un directorio que contiene información de configuración para el paquete, como las dependencias del paquete.
- *“Results_images”* Es un directorio que contiene las imágenes de los resultados de la detección de plagio.
- *“Unit_test”* un directorio que contiene el archivo *"unit_tests.py"* que contiene las pruebas unitarias que se utilizaron para probar la funcionalidad del proyecto.
- *"README.md"* es el archivo de texto que contiene información sobre cómo utilizar el programa y los requisitos necesarios para ejecutarlo.
- *“requirements.txt”* Es un archivo que especifica las dependencias del proyecto.
- *“Results.md”* Es un archivo que describe los resultados de la detección de plagio.

- "run.py" Es el archivo de Python principal del proyecto en Python, ya que se utiliza para ejecutar el programa de detección de plagio.

- *"check_plagiarism.py*" Es el archivo Python donde se realiza la detección de plagio con las técnicas empleadas para ello y se genera la salida. Contiene la lógica principal de detección de plagio.

- *“setup.py”* Es el archivo de Python que contiene información de configuración para el empaquetado y distribución del paquete.

El proyecto tiene como objetivo detectar plagio en documentos mediante el cálculo de la similitud coseno entre los documentos. Está dividido en los siguientes directorios:

El directorio *“Algorithms”* contiene los archivos de *AUC.py* que calcula el área bajo la curva (AUC) de una curva ROC (receiver operating characteristic). La curva ROC evalúa los clasificadores binarios, y la AUC mide el rendimiento del clasificador. Importa tres variables del archivo *check_plagarism.py* los cuales son *read_documents() , suspicious_files* y *original_files*. Estas variables se utilizan para obtener los documentos sospechosos y originales para realizar la detección de plagio. De igual manera, define manualmente las etiquetas de nombre y las etiquetas verdaderas de los documentos en unas lista llamadas *labels_name* y *true_labels* respectivamente. Las etiquetas verdaderas se establecen en 0 para

los archivos originales y en 1 para los archivos sospechosos de plagio.También se define la función *check_files(),* que utiliza *read_documents()* obteniendo la matriz de similitud entre los documentos originales y los sospechosos de plagio. La función recorre cada archivo sospechoso y cada archivo original verificando si la similitud entre ellos es mayor que 0.30. Si es así, agrega el nombre del archivo sospechoso a una lista de archivos sospechosos de plagio. Luego, la función devuelve la lista de archivos sospechosos de plagio. Posteriormente, se define la función *get_predictions(),* que llama a la función *check_files()* para obtener la lista de archivos sospechosos de plagio. La función luego recorre la lista de etiquetas de nombre y verifica si cada nombre de archivo está en la lista de archivos sospechosos de plagio. Si el nombre está en la lista, la función agrega un 1 a la lista de predicciones, de lo contrario, agrega un 0 a la lista de predicciones. La función devuelve la lista de predicciones. Utiliza las las variables *suspicious_files* y *original_files* que son definidas en el archivo *check_plagarism.py* ya que contienen las listas de nombres de archivos de los documentos originales y sospechosos, respectivamente. El archivo *AUC.py* utiliza estas listas para comparar todos los documentos sospechosos con todos los documentos originales y obtener una matriz de similitud. Esta matriz se utiliza para determinar si un documento sospechoso es plagio o no en relación con los documentos originales. Por lo tanto, el archivo *AUC.py* depende de las funciones y variables del archivo *check_plagarism.py*. Finalmente,se llama a *get_predictions()* para obtener la lista de predicciones y luego llama a *roc_auc_score()* de la biblioteca *scikit-learn* para calcular el AUC de la curva ROC. El resultado se imprime en la consola. Del archivo *check_plagarism.py* se obtiene

El directorio *Unit_test* contiene el archivo *unit_tests.py* contiene un conjunto de pruebas unitarias para evaluar la función *compare_files* del archivo *check_plagarism.py.* La función *unit_test_compare_files* llama a la función *compare_files* con dos documentos de prueba y un valor de *ngram* especificado (trigramas), y devuelve el resultado de la similitud calculada por *compare_files.* El objetivo de estas pruebas es asegurarse de que la función *compare_files* funciona correctamente antes de integrarse en el sistema de detección de plagio.

El archivo *check_plagarism.py* contiene funciones para comparar documentos y medir su similitud, utilizando técnicas de procesamiento de lenguaje natural y aprendizaje automático. En este módulo, se importan las librerías necesarias como *os* que permite acceder a los archivos en el sistema obteniendo los nombres de los archivos en los directorios del proyecto que contienen los archivos a analizar y comparar, *numpy* el cual se utiliza para crear una matriz de similitud, que guarda la similitud de coseno entre cada par de documentos originales y sospechosos*, joblib* que paraleliza la ejecución del proceso de comparación de los archivos, por lo que aumenta la eficiencia del proyecto*, nltk* que realiza la tokenización de palabras en los documentos, lo que permite la construcción de vectores de palabras*, TfidfVectorizer* de la librería *sklearn.feature_extraction.text* el cual se utiliza para convertir los documentos de texto a vectores de características basados en el método TF-IDF, que da más peso a las palabras que aparecen con menos frecuencia en los documentos, y *cosine_similarity* de la librería *sklearn.metrics.pairwaise* que se utiliza para calcular la similitud de coseno entre los vectores de características de dos documentos. De igual forma se emplea el modelo *punkt* de la librería *nltk* el cual es un tokenizador que por medio de la función *word_tokenize* divide los textos en oraciones y palabras individuales, siendo necesaria para poder vectorizar los textos y compararlos usando la medida de similitud coseno en la función *compare_files()*. La función *compare_files* utiliza un vectorizador de términos-frecuencia inversa de documentos (TF-IDF) para representar cada documento en forma de vectores.Después, se calcula la similitud de coseno entre los vectores que representan los dos documentos a comparar. El parámetro *ngram_range* de *TfidfVectorizer*

define el rango de n-gramas utilizados en el vectorizador, se utiliza un rango de trigramas,por lo que los n-gramas tienen una longitud de tres palabras. Cabe mencionar que se decidió emplear los trigramas en vez de bigramas o unigramas debido a que permite capturar combinaciones de tres palabras que pueden ser características distintivas de un documento ya que la cantidad de texto en los archivos aunque no es muy corta de igual forma puede ser considerada un poco extensa. La función *read_documents* lee todos los documentos que se encuentran en las carpetas y calcula las similitudes entre ellos. Los resultados se guardan en una matriz de similitudes y regresa el valor obtenido de esas similitudes.

El archivo *run.py* se encarga de imprimir los resultados de la detección de plagio. Para ello, se importan las variables *similarities*, *suspicious_files* y *original_files* desde el archivo *el a*. Luego, se itera sobre cada par de documentos comparados, y si el puntaje de similitud coseno es mayor que 0.3, se imprime un mensaje de alerta que indica que se ha detectado plagio. Este valor de umbral es ajustable, y se puede modificar en el código. De igual forma, *run.py* se encarga de ejecutar el proyecto dando el resultado de plagio en porcentaje, siendo mostrados solamente los archivos sospechosos que tuvieron al menos un 30% de similitud con los archivos originales.

En este modelo, se utilizan la técnica de N-gramas en conjunto con la técnica de similitud del coseno porque resulta ser más efectivo para detectar plagio. La técnica de N-gramas se utiliza para convertir el texto en una representación numérica, permitiendo que el algoritmo de similitud del coseno compare la similitud entre dos textos. El uso de N-gramas captura las relaciones entre las palabras y su orden, lo que resulta útil en la detección de plagio. Por otro lado, la técnica de similitud del coseno mide la similitud geométrica entre los dos vectores numéricos, que son la representación de los textos en términos de N-gramas. La técnica de similitud del coseno es efectiva porque no solo tiene en cuenta la presencia de las palabras, sino también su frecuencia relativa en el documento.

En este caso, se decidió utilizar trigramas en lugar de bigramas o unigramas porque pueden capturar mejor la estructura y el contexto de una oración o frase. Al usar trigramas, se pueden identificar combinaciones específicas de tres palabras que pueden ser únicas para un documento o texto, lo que es útil para identificar similitudes en el uso de lenguaje o estilo de escritura. Además, los trigramas pueden capturar secuencias de palabras más largas y complejas que los bigramas o los unigramas, lo que puede resultar en una mayor precisión en la detección de similitudes.

Después de realizar estos cambios, se pudo comparar un documento sospechoso o plagiado con varios documentos originales, mostrando cada resultado de plagio sin que se compararan documentos originales con originales o sospechosos con sospechoso entre sí. Además, los resultados se ejecutaban de una manera mucho más rápida y estos resultados al comprobar con los archivos plagiados si efectivamente aparecieron con un porcentaje exacto de plagio se pudo ver que el detector de plagio resultó ser efectivo. En resumen, la combinación de N-gramas con la técnica de similitud del coseno y el uso de trigramas resultó ser la mejor opción para detectar plagio de manera precisa y efectiva.

Explicación del código por partes:

1) Del directorio *Algorithms* del proyecto:

   El archivo *AUC.py:*

```
from sklearn.metrics import roc_auc_score

from check_plagarism import read_documents, suspicious_files,
original_files
```

Importa la función *roc_auc_score* del módulo *sklearn.metrics*que se utiliza para calcular el UC-ROC (Area Under the Receiver Operating Characteristic Curve) de un modelo de clasificación binaria. También se importan las variables *read_documents, suspicious_files* y *original_files* del archivo *check_plagiarism* , que se utilizan en el cálculo del AUC. La variable *read_documents* obtiene la matriz de similitud entre los documentos originales y los documentos sospechosos, mientras que *suspicious_files* y *original_files* son listas que contienen los nombres de los archivos sospechosos y originales, respectivamente.

```
# Manually define the labels names

labels_name = ['FID-01', 'FID-02', 'FID-03', 'FID-04', 'FID-05',
'FID-06', 'FID-07', 'FID-08', 'FID-09', 'FID-10', 'FID-11', 'FID-12',
'FID-13', 'FID-14','FID-15']

# Manually define the true labels (0 for original, 1 for suspicious)

true_labels = [1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1]
```

Se definen manualmente los nombres de las etiquetas correspondientes para cada archivo en el conjunto de datos. En este caso, se está utilizando una lista de nombres de etiquetas *labels_name,* que contiene 15 elementos, cada uno correspondiente a un archivo en el conjunto de datos en los subdirectorios. Además, se define la lista *true_labels* que contiene las etiquetas verdaderas para cada archivo en el conjunto de datos. Las etiquetas verdaderas se definen como 0 para los archivos originales y 1 para los archivos sospechosos. En este caso, hay 6 archivos originales y 9 archivos sospechosos, por lo que hay 6 etiquetas verdaderas 0 y 9 etiquetas verdaderas 1 en la lista *true_labels*.

```
# Function that checks if similarity scores is > than 0.30, if so,
append the plagarisim file name

def check_files():

    # Get the similarity matrix between all suspicious and original
documents

    similarities = read_documents()

    plag_docs_pred = []
```

```
    # Output the documents with similarity scores above 0.30

    for i, plagiarized_file in enumerate(suspicious_files):

        for j, original_file in enumerate(original_files):

            if similarities[i, j] > 0.30:

                # If so, append plagiarized_file name to plag_docs_pred

                if plagiarized_file.split('.')[0] not in
plag_docs_pred:

                    # If plagiarized_file name is already in
plag_docs_pred, it doesn't append it twice


plag_docs_pred.append(plagiarized_file.split('.')[0])

    return plag_docs_pred
```

Se define la función *check_files(),* que se encarga de verificar si los archivos sospechosos están plagiados o no en base a los puntajes de similitud calculados previamente en el archivo *check_plagiarism.py*. Primero, se llama a la función *read_documents()* que lee todos los documentos y obtiene la matriz de similitud calculada entre los documentos sospechosos y los documentos originales.Luego, se itera sobre cada archivo sospechoso y se compara con cada archivo original. Si el puntaje de similitud es mayor a 0.30, se considera que el archivo sospechoso está plagiado del archivo original correspondiente. El nombre del archivo sospechoso se agrega a una lista llamada *plag_docs_pred* solo si no está ya presente en la lista. Finalmente, la función devuelve la lista de archivos sospechosos que se consideran plagiados en base al criterio de similitud establecido.

```


#Function that gets the predictions given plag_docs_pred of
"check_files()"

def get_predictions():

    predictions = []

    plag_docs_pred = check_files()

    # Check if the plag_docs_pred names are in labels_name

    for name in labels_name:
```

```
        # If so, append 1 to predictions empty list
        if name in plag_docs_pred:
            predictions.append(1)
        # If not, append 0 to predictions empty list
        else:
            predictions.append(0)
    # Returns the list with the predictions
    return predictions


predictions = get_predictions()
# Give AUC score
auc = roc_auc_score(true_labels, predictions)
print('----------------------------------------')
print(f"El Area bajo la curva AUC es: {auc:.2f}")
print('----------------------------------------')
```

Se define la función *get_predictions()* que utiliza la función *check_files()* para obtener una lista de documentos plagiados, y luego verifica si los nombres de estos documentos están presentes en la lista *labels_name.* Para cada nombre que se encuentre, la función agrega 1 a la lista *predictions,* de lo contrario, agrega 0. Finalmente, la función devuelve la lista de predicciones. Luego se llama a *get_predictions()* para obtener la lista de predicciones, y luego calcula el puntaje AUC utilizando la función *roc_auc_score* del módulo *sklearn.metrics*. Las etiquetas verdaderas se dan mediante la lista *true_labels* que se definió manualmente anteriormente. El puntaje AUC se imprime en la consola.

2) En el archivo *check_plagarism.py* del proyecto:

```
# The necessary libraries are imported
import os  # For file management
import numpy as np  # For numerical computations
from joblib import Parallel, delayed  # For parallel processing
```

```
from nltk.tokenize import word_tokenize  # For natural language processing

from sklearn.feature_extraction.text import TfidfVectorizer  # For machine
learning

from sklearn.metrics.pairwise import cosine_similarity

import nltk

# Pre-trained sentence tokenizer to split text into individual sentences

nltk.download('punkt')
```

En esta sección se importan las librerías necesarias para la detección de plagio utilizando la técnica de N-gramas en conjunto con la técnica de similitud del coseno.

La librería *os* se utiliza para la gestión de archivos, permitiendo acceder a los archivos necesarios para la comparación de textos.

La librería *numpy* se utiliza para realizar cálculos numéricos, lo que permite la representación de los textos en forma de vectores numéricos para poder compararlos con la técnica de similitud del coseno.

La función *Parallel* y *delayed* de la librería *joblib* se utilizan para procesamiento paralelo, lo que acelera el proceso de comparación de los documentos.

La librería *nltk* se utiliza para procesamiento del lenguaje natural, en este caso, para realizar la tokenización de palabras en los documentos.

La librería *TfidfVectorizer* de la librería *sklearn* se utiliza para convertir los documentos en vectores numéricos utilizando la técnica de TF-IDF, que asigna un valor numérico a cada palabra en función de su frecuencia en el documento y en los documentos totales.

Finalmente, se utiliza la función *cosine_similarity* de la librería *sklearn* para calcular la similitud del coseno entre los vectores numéricos de los documentos, lo que permite determinar la similitud entre los textos.

Se descarga el tokenizador de oraciones pre-entrenado *punkt* de la librería *nltk,* el cual se utiliza para separar el texto en oraciones individuales antes de realizar la tokenización de palabras, lo que permite una mejor representación del texto en forma de vectores numéricos.

```
# Two directories are specified

# For the original documents

original_route = "./Detection_Files/Originals"

# For the suspicious documents

suspicious_route = "./Detection_Files/Suspicious"
```

```
# The list of original and suspicious files is obtained

original_files = os.listdir(original_route)

suspicious_files = os.listdir(suspicious_route)
```

Se especifican dos directorios: uno para los documentos originales y otro para los documentos sospechosos. Estos directorios son *original_route* y *suspicious_route,* respectivamente. Aunque si el usuario deseara comparar los documentos plagiados para poder comprobar la efectividad del detector de plagio simplemente se tendrá que cambiar la ruta del directorio de los documentos sospechosos por la ruta del directorio de los archivos plagiados *Plag*.

Luego, se utiliza la función *os.listdir* para obtener una lista de los nombres de archivo en cada directorio. Esta función devuelve una lista con los nombres de todos los archivos en el directorio especificado. Se obtiene una lista de los nombres de archivo en el directorio de documentos originales y otra lista en el directorio de documentos sospechosos.

Estas listas se guardan en las variables *original_files* y *suspicious_files,* respectivamente. Almacenar los nombres de archivo en estas listas permite que el código itere a través de cada archivo en cada directorio para realizar la comparación de plagio.

```
# Function that takes two documents as input and returns their cosine
similarity score

def compare_files(doc1, doc2, ngram_range):

  # Vectorizer extracts all possible 3-grams (sequences of 3 consecutive
words) from the text

    tfidf_vectorizer = TfidfVectorizer(ngram_range=(

        ngram_range, ngram_range), tokenizer=word_tokenize)  # Generate
document vectors

    tfidf_doc1 = tfidf_vectorizer.fit_transform([doc1])

    tfidf_doc2 = tfidf_vectorizer.transform([doc2])

    return cosine_similarity(tfidf_doc1, tfidf_doc2)[0][0]
```

Se define una función llamada *compare_files ()* que toma dos documentos como entrada y devuelve su puntaje de similitud del coseno. La función utiliza un vectorizador de términos de frecuencia de documentos inversa (TF-IDF) para generar vectores de documentos para los dos documentos de entrada. Se establece el rango de N-gramas utilizando el parámetro *ngram_range*, que se pasa como argumento a la función. Se utiliza el rango de N-gramas de trigramas o 3 palabras consecutivas del texto.

El vectorizador TF-IDF utiliza la función *word_tokenize* de la biblioteca *nltk* para dividir el texto en palabras individuales y luego extrae todos los posibles N-gramas de longitud *ngram_range* (siendo de 3) de las palabras individuales. Luego, se utiliza la función *fit_transform* del vectorizador para generar un vector de características TF-IDF para el primer documento y la función *transform* para generar el vector de características para el segundo documento.

Finalmente, la función *cosine_similarity* calcula la similitud del coseno entre los vectores de características de los dos documentos. El puntaje de similitud se devuelve como la salida de la función.

```
def read_documents():

    # Performs pairwise document comparison between all suspicious and
original documents and generates

    #  a similarity matrix containing the similarity scores between each pair
of documents

    similarities = np.zeros((len(suspicious_files), len(original_files)))

    # The first loop iterates over each suspicious document, and the second
loop iterates over each original document

    # For each pair of documents, the text content of each document is read in
and passed to the  function to calculate the similarity score

    for i, plagiarized_file in enumerate(suspicious_files):

        sus_route = os.path.join(suspicious_route, plagiarized_file)

        with open(sus_route, 'r', encoding="utf-8") as plagiarized_file:

            plagiarized_text = plagiarized_file.read()

        for j, original_file in enumerate(original_files):

            org_route = os.path.join(original_route, original_file)

            with open(org_route, 'r', encoding="utf-8") as original_file:

                original_text = original_file.read()

                # The similarity score is then stored in the array at the
corresponding location

            similarities[i, j] = compare_files(
```

```
                original_text, plagiarized_text, 3)

    return similarities
```

La función *read_documents()* realiza una comparación de documentos. La función realiza un bucle anidado para comparar cada documento sospechoso con cada documento original. La matriz de similitud se inicializa como una matriz de ceros con dimensiones *(len(suspicious_files)), (len(original_files))*.

En el primer bucle *for,* cada documento sospechoso se abre y se lee el contenido del texto en la variable *plagiarized_text.* En el segundo bucle *for,* se itera sobre cada documento original. Para cada par de documentos, se abre y se lee el contenido del texto la variable *original_text.* La función *compare_files* se llama con *original_text*, *plagiarized_text* y el rango de N-gramas de 3.

El resultado de cada comparación se almacena en la matriz *similarities.* Al final se devuelve la matriz *similarities*.

3) En el archivo *run.py* del proyecto:

```
from check_plagarism import similarities, suspicious_files, original_files
```

Se importan las variables desde el archivo *check_plagarism.py* las cuales son *similarities, suspicious_files* y *original_files.*

```
if __name__ == '__main__':

    # The results are printed out for each pair of documents

    for i, plagiarized_file in enumerate(suspicious_files):

        for j, original_file in enumerate(original_files):

            # If the similarity score between an original and a suspicious
document is greater than 30%

            if similarities[i, j] > 0.30:

                # The output includes the name of the original and suspicious
document, as well as the similarity score

                print(f"¡Alerta! Se ha detectado plagio en el documento
{plagiarized_file}. \nLa similitud de coseno entre el documento original
{original_file} y el documento plagiado {plagiarized_file} es:
{similarities[i,j]*100:.2f}%")
```

```
print("----------------------------------------------------
------------------------------------------------------------
```

```
-------------------------------------------------------------
-------")
```

Esta parte del proyecto se ejecuta cuando se ejecuta el archivo. Itera sobre cada par de documentos (un documento sospechoso y un documento original), y si la puntuación de similitud (obtenida anteriormente en el archivo *check_plagarism.py*) entre los dos documentos es mayor al 30%, se imprime un mensaje de alerta indicando que se ha detectado plagio en el documento sospechoso, junto con el nombre del documento original y la puntuación en porcentaje de similitud entre ambos documentos.

Se decidió que el porcentaje de similitud que se considerara plagio fuese mayor al 30% debido a que al momento de realizar las pruebas unitarias el porcentaje que nos dió mejores resultados en la exactitud de la detección de plagio fué de 0.30.

4) En el archivo *requirements.txt* del proyecto:

```
click==8.1.3

colorama==0.4.6

joblib==1.2.0

nltk==3.8.1

numpy==1.24.3

-egit+https://github.com/RamirezFernanda/Plagarism_detection.git@253588bacccb3
fa2d8eeb868a6abec709b142b3c#egg=plagiarism_detection

regex==2023.3.23

scikit-learn==1.2.2

scipy==1.10.1

threadpoolctl==3.1.0

tqdm==4.65.0
```

Contiene una lista de paquetes de Python y sus versiones correspondientes que se necesitan para ejecutar el proyecto. A continuación se muestra la lista de paquetes incluidos en el archivo:

- click==8.1.3
- colorama==0.4.6
- joblib==1.2.0
- nltk==3.8.1
- numpy==1.24.3
- plagirism_detection (descargado desde un repositorio de Github específico)
- regex==2023.3.23

- scikit-learn==1.2.2
- scipy==1.10.1
- threadpoolctl==3.1.0
- tqdm==4.65.0

Estos paquetes se pueden instalar fácilmente utilizando la herramienta *pip* de Python, utilizando el archivo *requirements.txt* como entrada.

Pruebas Unitarias:

Del directorio *Unit_test*:

En el archivo *unit_tests.py* del proyecto:

```
from sklearn.metrics.pairwise import cosine_similarity

import os  # For file management

import numpy as np  # For numerical computations

from joblib import Parallel, delayed  # For parallel processing

from nltk.tokenize import word_tokenize  # For natural language processing

from sklearn.feature_extraction.text import TfidfVectorizer  # For machine
learning

import nltk
```

Se importan las librerías necesarias como en el archivo *check_plagarism.py*.

```
# Two directories are specified

# For the original documents

original_route = "./Detection_Files/Originals"

# For the suspicious documents

suspicious_route = "./Detection_Files/Suspicious"



# The list of original and suspicious files is obtained

original_files = os.listdir(original_route)

suspicious_files = os.listdir(suspicious_route)
```

Se especifican los dos directorios, *original_route* y *suspicious_route*, que corresponden a las rutas de los archivos originales y sospechosos, respectivamente.

Se utiliza la función os.listdir() para obtener una lista de los nombres de los archivos en cada directorio, que se almacenan en las variables *original_files* y *suspicious_files*.

```
print(

    f'Original files found in the path {original_route}: {original_files} \n
-------------- \nTotal files: {len(original_files)}\n --------------')

print(

    f'Suspicious files found in the path {suspicious_route}:
{suspicious_files} \n -------------- \nTotal files: {len(suspicious_files)}\n
--------------')
```

Se imprimen los nombres de los archivos encontrados en cada directorio, así como el número total de archivos encontrados en cada uno.

```
def compare_files(doc1, doc2, ngram_range):

  # Uses N-grams to extract features from the text data

  # Vectorizer will extract all possible 3-grams (sequences of 3 consecutive
words) from the text

    tfidf_vectorizer = TfidfVectorizer(ngram_range=(

        ngram_range, ngram_range), tokenizer=word_tokenize)  # Generate
document vectors

    tfidf_doc1 = tfidf_vectorizer.fit_transform([doc1])

    tfidf_doc2 = tfidf_vectorizer.transform([doc2])

    return cosine_similarity(tfidf_doc1, tfidf_doc2)[0][0]
```

La función *compare_files()* compara la similitud entre dos documentos usando la métrica de similitud de coseno. Toma como entrada dos documentos y el rango de N-gramas que extrae las características de los datos de texto. . La función crea un vectorizador TF-IDF con el rango de N-gramas (trigramas) y tokeniza los documentos usando la función *word_tokenize* de *nltk.* Después, se genera un vector de características para cada documento utilizando el vectorizador TF-IDF. Finalmente, la función devuelve la similitud de coseno entre los vectores de características de los dos documentos, lo que representa la similitud entre los dos documentos.

```
def unit_test_compare_files(doc1, doc2, ngram):

    similarity_score = compare_files(doc1, doc2, ngram)

    return similarity_score



print(f'La similitud para la prueba unitaria
es:\n{unit_test_compare_files("/content/drive/MyDrive/Copia de
construccion/documentos-genuinos/org-001.txt","/content/drive/MyDrive/Copia de
construccion/documentos-con texto de otros/plg-001.txt",1)}')
```

Define la función *unit_test_compare_files()* que toma dos argumentos que son el contenido de dos documentos y *ngram* que indica el rango de los n-gramas utilizados para extraer las características de los documentos.

La función utiliza la función *compare_files* definida anteriormente para comparar la similitud entre los documentos utilizando el rango de n-gramas. Luego, devuelve la puntuación de similitud como resultado.

Finalmente, la función *unit_tests_compare_files* es llamada en el archivo *run.py* utilizando dos rutas de archivo diferentes como argumentos para los documentos. El resultado de la comparación de similitud se imprime en la consola.

Liga del Repositorio:

https://github.com/RamirezFernanda/Plagarism_detection.git