Detector de Similitud de Texto

TC2004B – Análisis de Ciencia de Datos
Tecnológico de Monterey

Integrantes del Equipo:

Diego Alejandro Balderas Tlahuitzo, A01745336

March 2024

Contenido

Contenido

Resumen ejecutivo

Este proyecto se centra en el desarrollo de un sistema avanzado para la detección de plagio utilizando técnicas de procesamiento de lenguaje natural y aprendizaje automático. A través de dos enfoques distintos, el sistema busca no solo identificar el plagio textual directo, sino también discernir formas más sutiles como el parafraseo y la reelaboración del contenido. El uso de modelos semánticos y algoritmos de comparación estructural permite al sistema detectar y clasificar diferentes tipos de plagio con una precisión significativa. La evaluación del sistema revela una alta capacidad para identificar documentos plagiados y un profundo entendimiento de las diversas técnicas de manipulación de texto, subrayando la efectividad de combinar métodos tradicionales con tecnologías avanzadas de inteligencia artificial.

1 Introducción

El uso del término "plagio" tiene sus raíces en la antigua Roma, donde la palabra latina "plagiarius" se usaba para describir a alguien que secuestraba a personas o robaba bienes. Sin embargo, su aplicación al robo literario o intelectual comenzó con el poeta romano Martial, quien acuñó este término para denunciar a otro poeta que recitaba sus obras como si fueran propias. Aunque la disputa de Martial fue una de las primeras instancias documentadas del concepto de plagio relacionado con la autoría, fue significativamente diferente de nuestro entendimiento actual. En lugar de enfocarse en el crédito creativo o la originalidad, el enfoque estaba en la compensación económica por el trabajo creativo, algo que refleja las actitudes de la época hacia la propiedad intelectual[1].

La perspectiva sobre la propiedad intelectual comenzó a cambiar durante el Renacimiento y, más tarde, en la Ilustración, cuando se inició la formalización del concepto de derechos de autor. A medida que avanzaba el siglo XVIII, las actitudes hacia el plagio se endurecieron y surgió la idea moderna de "plagio", donde se esperaba que los autores crearan obras originales. La noción de creatividad y propiedad intelectual comenzó a tomar forma, y el acto de plagiar pasó a ser visto cada vez más como una infracción grave contra la autoría y la originalidad[2].

En el aspecto tecnológico, los primeros verificadores de plagio en línea surgieron a finales de los años 90 con el crecimiento de Internet, donde herramientas como Turnitin empezaron a aparecer para comparar el contenido presentado con una vasta base de datos de documentos académicos, sitios web y publicaciones. Los algoritmos de coincidencia de texto se desarrollaron en la década de 2000 para analizar el texto en busca de similitudes, lo que significó un gran paso adelante en la capacidad de detectar el plagio de manera más sistemática y eficiente[3].

El avance más reciente en la detección de plagio ha sido la incorporación de inteligencia artificial y aprendizaje automático, que ha mejorado la precisión en la identificación de formas sutiles de plagio, incluso el contenido parafraseado o reescrito. Hoy en día, estas herramientas no sólo son precisas en la detección de texto, sino que algunas también pueden identificar plagio en contenido multimedia, lo que amplía el alcance para mantener la originalidad y los derechos de propiedad intelectual en diversos medios digitales[3].

1.1 Descripción del Problema

En el entorno académico y profesional actual, el avance tecnológico ha revolucionado la manera en que generamos y accedemos al conocimiento. Sin embargo, este progreso también ha traído consigo desafíos significativos; uno de los más prominentes es la gestión del plagio. La facilidad con la que se puede obtener información en línea ha llevado a un aumento en la incidencia del plagio, complicando la tarea de mantener la integridad académica y los derechos de autor.

Aunque existen múltiples herramientas para detectar el plagio, muchas dependen de la identificación de coincidencias textuales directas y pueden no ser suficientemente efectivas frente a métodos más sofisticados de copia, como el parafraseo inteligente o la ocultación deliberada de la copia. Estas herramientas tradicionales también pueden fallar en adaptarse al volumen y la diversidad de datos que ahora están disponibles, desde publicaciones académicas hasta contenido digital informal.

En este contexto, surge la necesidad de desarrollar una solución de detección de plagio que no solo aborde el plagio textual directo, sino que también tenga la capacidad de discernir entre el parafraseo y la reelaboración de contenido. La herramienta ideal debería aprovechar las técnicas avanzadas de procesamiento de lenguaje natural y aprendizaje automático para ofrecer

una evaluación más profunda y matizada del contenido, asegurando así la originalidad y la autenticidad del trabajo intelectual en una era definida por el acceso a información ilimitada.

1.2 Descripción de los datos

En este proyecto se utilizó un conjunto de datos inicial compuesto por 110 documentos originales, los cuales son abstracts de artículos científicos del año 2024, abarcando diversas áreas de la inteligencia artificial. Este conjunto de datos proporciona una base sólida y representativa para la detección de plagio, dado que incluye terminología y estructuras específicas de un campo altamente técnico, importante señalar que todos estos documentos están escritos en el idioma inglés.

Además, como parte del reto inicial, se proporcionaron 10 documentos adicionales que sirven como ejemplos explícitos de distintos tipos de plagio. Estos ejemplos fueron cruciales para entender y modelar las diversas formas en que el plagio puede manifestarse en textos académicos. Los tipos de plagio abordados incluyen:

1. **Inserción o reemplazo de frases**: Modificación de partes del texto original mediante la sustitución o adición de nuevas frases o frases de otro(s) documento(s).
2. **Desordenar las frases**: Alteración del orden original de las frases para disimular el plagio.
3. **Cambio de tiempo**: Modificación de los verbos para cambiar su tiempo gramatical.
4. **Cambio de voz**: Transformación de la voz activa en pasiva, o viceversa.
5. **Parafraseo**: Reescritura del texto original utilizando diferentes palabras y estructuras sin cambiar el significado.

Para simular y generar casos de entrenamiento más complejos, se desarrollaron funciones en Python que aplican estas técnicas de plagio a los 110 documentos originales, creando variaciones con diferentes porcentajes de contenido plagiado. Este proceso permitió la creación de un conjunto de datos robusto para el entrenamiento y la validación de los modelos, asegurando que son capaces de identificar plagio en condiciones variadas y con diferentes grados de dificultad.

Finalmente, se recibió un conjunto de prueba compuesto por 30 documentos. Estos documentos fueron utilizados como la evaluación final para nuestra herramienta de detección de plagio, permitiendo medir la eficacia del sistema en un entorno controlado y desconocido previamente para el modelo.

2 Estado del Arte

2.1 Cómo se aborda este problema

La detección de plagio ha evolucionado considerablemente con el desarrollo de tecnologías en procesamiento de lenguaje natural (NLP) y aprendizaje automático (ML). Estas tecnologías permiten un análisis profundo y sofisticado del texto. A continuación, exploraremos dos enfoques principales, destacando sus fortalezas y debilidades en el contexto de la detección de plagio.

Preprocesamiento de Texto:

El preprocesamiento es esencial para preparar los datos para análisis efectivos y precisos. Los pasos incluyen:

- **Stemming y Lemmatización:** Estos procesos reducen las palabras a sus raíces o formas base, donde el stemming puede generar formas inexactas y la lemmatización ofrece mayor precisión al considerar el contexto gramatical.
- **Eliminación de Ruido:** Remueve caracteres no alfabéticos y stopwords, limpiando los textos de elementos irrelevantes para los análisis

2.1.1 Enfoque 1: Métodos Tradicionales y Análisis Estructural

Este enfoque se centra en el uso de n-grams y `CountVectorizer`, que son fundamentales para el análisis estructural del texto.

- **N-Grams:** Permiten la captura de secuencias de palabras, esenciales para identificar plagios que involucran cambios sutiles en el texto. Aunque son efectivos para detectar coincidencias exactas, pueden no capturar manipulaciones más sutiles en textos largos.
- **CountVectorizer**: Transforma el texto en una matriz de frecuencias de términos, útil para análisis estructurales básicos, pero puede no ser suficiente para diferenciar contextos semánticos complejos.
- **Similitud de Cosine**: Utilizada para evaluar la proximidad entre documentos, aplicable en ambos enfoques para determinar el porcentaje de similitud.

2.1.2 Enfoque 2: Inteligencia Artificial y Modelos Semánticos

Incorpora técnicas avanzadas de inteligencia artificial y vectorización TF-IDF para un análisis más profundo y contextual del texto.

- **Modelos de Machine Learning**: Utilizan estructuras como los transformers para analizar el contexto completo del texto, lo que los hace altamente efectivos en la detección de formas sofisticadas de plagio.
- **TF-IDF**: Además de `CountVectorizer`, este enfoque utiliza TF-IDF para transformar textos en vectores numéricos que resaltan la importancia de las palabras, permitiendo análisis más finos de las similitudes textuales.
- **Similitud de Cosine**: Crucial para evaluar la similitud entre documentos vectorizados, esencial para detectar coincidencias no obvias.

2.1.3 Validación y Optimización

Se realizan pruebas rigurosas en la herramienta para validar su precisión y eficacia, ajustando continuamente los algoritmos e hiperparametros (tamaño de ngrmas, arquitectura de Deep Learning, cantidad de épocas de entrenamiento, ajuste de capas de la arquitectura) para adaptarse a las necesidades cambiantes buscando mejorar la detección del porcentaje de plagio y tipo de plagio de cada documento.

2.1.4 Evaluación de Modelos en la Detección de Plagio

Para asegurar la efectividad y fiabilidad de los modelos de detección de plagio, es esencial utilizar métricas de evaluación robustas que permitan medir su desempeño de manera objetiva. Las métricas más comunes incluyen:

- **Precisión (Precision)**: Mide la exactitud de las predicciones positivas del modelo. La precisión es la proporción de verdaderos positivos (documentos correctamente identificados como plagiados) entre todos los casos clasificados como plagiados. Esta métrica es crucial para evaluar la calidad de las alertas de plagio, minimizando los falsos positivos.
- **Recall (Sensibilidad)**: También conocida como sensibilidad, mide la capacidad del modelo para identificar todos los casos relevantes dentro de un conjunto de datos (es decir, la proporción de verdaderos positivos entre la suma de verdaderos positivos y falsos negativos). Es especialmente importante en aplicaciones donde no detectar un incidente (como un caso de plagio) podría tener consecuencias graves.
- **Accuracy (Exactitud)**: Calcula el porcentaje de predicciones correctas (tanto verdaderos positivos como verdaderos negativos) en relación con el total de casos. Esta métrica proporciona una visión general de la efectividad del modelo en todos los aspectos de la detección.
- **F1 Score**: Combina la precisión y el recall en una única métrica mediante su media armónica. Es útil cuando las clases están desbalanceadas, proporcionando un balance entre la precisión y el recall para dar una perspectiva integral del rendimiento del modelo.
- **AUC (Área Bajo la Curva ROC)**: Representa la capacidad del modelo para discriminar entre clases. Un AUC de 1.0 indica un modelo perfecto, mientras que un AUC de 0.5 sugiere un rendimiento no mejor que el azar. Esta métrica es valiosa para evaluar la robustez del modelo en diferentes umbrales de clasificación.

El uso de estas métricas permite al equipo evaluar y comparar la efectividad de diferentes modelos y enfoques en la detección de plagio, asegurando que los sistemas implementados sean tanto precisos como confiables.

2.1.5 Selección de Técnicas de IA y Deep Learning

Dentro del campo de detección de plagio, la selección de técnicas específicas de inteligencia artificial y deep learning se fundamenta en su reconocida capacidad para analizar profundidades lingüísticas y semánticas del texto. Modelos avanzados como BERT y algoritmos de procesamiento de texto como TF-IDF fueron seleccionados por su potencial demostrado en contextos de análisis textual. BERT, en particular, ha sido ampliamente adoptado en una variedad de tareas de procesamiento de lenguaje natural debido a su eficacia en entender contextos complejos en el texto, una característica crucial para sistemas de detección de plagio que requieren un alto nivel de precisión y la capacidad de comprender sutilezas en el texto【45†source】.

Además, el uso de deep learning es crucial para identificar patrones complejos en los tipos de plagio, permitiendo realizar predicciones efectivas sobre documentos sospechosos. Las redes neuronales profundas, especialmente en configuraciones que combinan redes neuronales convolucionales (CNNs) y redes neuronales recurrentes (RNNs), han demostrado ser efectivas en la clasificación de documentos técnicos y académicos al aprender características intrincadas y patrones dentro de grandes volúmenes de datos【51†source】. Estas técnicas permiten una comprensión detallada del contenido textual y son adecuadas para detectar variaciones sutiles de texto que caracterizan el plagio sofisticado.

El deep learning, en particular, facilita el modelado de datos no lineales y la detección de características no evidentes a simple vista, aspectos esenciales para sistemas avanzados de detección de plagio. La elección de estas herramientas está justificada por su demostrada eficacia en tareas similares y por su capacidad para manejar complejidades textuales y semánticas que son típicas en casos de plagio.

3 Propuesta de Solución

En este proyecto, se desarrollaron dos enfoques distintos para la detección de plagio, cada uno utilizando diferentes técnicas y herramientas para abordar el problema. A continuación, se describen ambos enfoques y las funciones clave que los constituyen.

3.1 Enfoque 1: Basado en Ngrams

El primer enfoque se centra en el uso de n-grams para detectar similitudes textuales directas entre documentos. Este método es particularmente efectivo para identificar el plagio más evidente, donde se copian frases o párrafos enteros con pocos o ningún cambio.

Función de Importación y Preparación de Datos:

```
plagiarism_detection.py > ...
import os
import string
import time
import numpy as np
from nltk import download
from nltk.stem import PorterStemmer
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.metrics import roc_curve, auc

# Descarga de recursos necesarios de NLTK
inicio = time.time()
download('punkt')
download('stopwords')
```

Librerías y Descargas: Se importan librerías esenciales como NLTK para el procesamiento de texto, y Scikit-learn para la extracción de características y la evaluación de similitudes. Además, se descargan recursos necesarios como 'punkt' y 'stopwords' para el tokenizado y la eliminación de palabras comunes que no añaden valor semántico significativo al análisis.

```python
# Funciones de utilidad
def read_files_in_directory(directory):
    files_contents = []
    file_names = []
    for filename in os.listdir(directory):
        if filename.endswith('.txt'):
            file_path = os.path.join(directory, filename)
            try:
                with open(file_path, 'r', encoding='utf-8') as file:
                    files_contents.append(file.read())
            except UnicodeDecodeError:
                with open(file_path, 'r', encoding='windows-1252') as file:
                    files_contents.append(file.read())
            file_names.append(filename)
    return file_names, files_contents
```

Lectura de Documentos: Se implementa una función para leer archivos de texto de un directorio especificado. Esta función maneja diferentes codificaciones para asegurar que todos los documentos sean accesibles y legibles sin errores de codificación.

```python
def preprocess(text):
    """ Procesa el texto aplicando normalización, eliminación de stopwords y stemming. """
    text = text.lower()
    text = text.translate(str.maketrans('', '', string.punctuation))
    tokens = word_tokenize(text)
    stop_words = set(stopwords.words('english'))
    stemmer = PorterStemmer()
    tokens = [stemmer.stem(token) for token in tokens if token not in stop_words]
    return ' '.join(tokens)
```

Normalización y Limpieza: El texto de cada documento se convierte a minúsculas, se eliminan las puntuaciones, y se divide en palabras o tokens. Posteriormente, se eliminan las palabras comunes (stopwords) y se aplica stemming para reducir las palabras a sus raíces, facilitando la comparación de textos.

```python
def generate_vector_space_models(original_texts, suspicious_texts):
    """ Genera modelos de espacio vectorial para textos originales y sospechosos. """
    if not all(isinstance(text, str) for text in original_texts + suspicious_texts):
        raise TypeError("All inputs must be strings.")
    vectorizer = CountVectorizer(analyzer='word', ngram_range=(1, 3))
    # Combina todos los textos para crear un espacio vectorial común
    combined_texts = original_texts + suspicious_texts
    vectorizer.fit(combined_texts)  # Ajusta el vectorizador a todos los textos
    original_vectors = vectorizer.transform(original_texts)  # Transforma los textos originales
    suspicious_vectors = vectorizer.transform(suspicious_texts)  # Transforma los textos sospechosos
    return original_vectors, suspicious_vectors, vectorizer.get_feature_names_out()
```

Generación de Espacio Vectorial: Utilizando CountVectorizer, se transforma el texto en una matriz de frecuencia de n-grams (con n variando de 1 a 3). Esto permite representar los documentos como vectores en un espacio común, donde cada dimensión corresponde a un n-gram específico.

```python
def evaluate_performance(similarities, threshold, ground_truth, suspicious_filenames1):
    """
    Evalúa el rendimiento de la herramienta de detección de plagio.
    similarities: matriz de similitudes entre documentos sospechosos y originales.
    threshold: el umbral de similitud para considerar un documento como plagiado.
    ground_truth: dict con clave = nombre del documento sospechoso y valor = bool indicando si es plagiado.

    Retorna un dict con las métricas TP, FP, TN, FN.
    """
    TP = FP = TN = FN = 0
    for i, susp_filename in enumerate(suspicious_filenames1):
        is_plagiarized = ground_truth[susp_filename]
        # Considera el documento plagiado si alguna similitud supera el umbral.
        detected_as_plagiarized = any(sim > threshold for sim in similarities[i])

        if is_plagiarized and detected_as_plagiarized:
            TP += 1
        elif not is_plagiarized and detected_as_plagiarized:
            FP += 1
        elif is_plagiarized and not detected_as_plagiarized:
            FN += 1
        elif not is_plagiarized and not detected_as_plagiarized:
            TN += 1

    return {"TP": TP, "FP": FP, "TN": TN, "FN": FN}
```

```python
def generate_report(performance_metrics, similarities, ground_truth_labels):
    """
    Genera un informe de rendimiento basado en las métricas dadas.
    performance_metrics: dict con TP, FP, TN, FN.
    similarities: lista de valores de similitud entre documentos.
    ground_truth_labels: lista de etiquetas de verdad fundamental (0 para no plagiado, 1 para plagiado).
    """
    TP = performance_metrics["TP"]
    FP = performance_metrics["FP"]
    TN = performance_metrics["TN"]
    FN = performance_metrics["FN"]
    precision = TP / (TP + FP) if TP + FP > 0 else 0
    recall = TP / (TP + FN) if TP + FN > 0 else 0
    accuracy = (TP + TN) / (TP + FP + TN + FN) if TP + FP + TN + FN > 0 else 0
    f1_score = 2 * (precision * recall) / (precision + recall) if precision + recall > 0 else 0

    # Calcular AUC
    fpr, tpr, thresholds = roc_curve(ground_truth_labels, similarities)
    roc_auc = auc(fpr, tpr)

    report = (
        f"True Positives: {TP}\n"
        f"False Positives: {FP}\n"
        f"True Negatives: {TN}\n"
        f"False Negatives: {FN}\n"
        f"Precision: {precision:.2f}\n"
        f"Recall: {recall:.2f}\n"
        f"Accuracy: {accuracy:.2f}\n"
        f"F1 Score: {f1_score:.2f}\n"
        f"AUC (ROC): {roc_auc:.2f}\n"
    )
    print(report)
    # Puedes añadir código aquí para guardar el informe en un archivo si es necesario.
```

Reporte de Rendimiento: Estas funciones generan un reporte detallado que incluye las métricas de evaluación como precisión, recall, F1 score, y AUC, utilizando las métricas de

verdaderos positivos, falsos positivos, verdaderos negativos y falsos negativos obtenidos de la función de evaluación.

```
    path_to_originals = './TextosOriginales'
    path_to_suspicious = './FinalTesting'
    # Cargar y procesar los documentos
    original_filenames, original_texts = read_files_in_directory(path_to_originals)
    suspicious_filenames, suspicious_texts = read_files_in_directory(path_to_suspicious)
    # Preprocesar todos los textos
    processed_originals = [preprocess(text) for text in original_texts]
    processed_suspicious = [preprocess(text) for text in suspicious_texts]
    # Crear modelos de espacio vectorial común
    original_vectors, suspicious_vectors, feature_names = generate_vector_space_models(processed_originals, processed_suspicious)
    # Calcular similitud de coseno
    similarities = cosine_similarity(suspicious_vectors, original_vectors)
    # Reportar resultados
    threshold = 0.0  # Umbral de similitud
    # Información de ground truth para evaluación
    ground_truth = {···
    # Aplanar las similitudes y preparar las etiquetas de ground truth
    all_similarities = []
    ground_truth_labels = []
    for i, filename in enumerate(suspicious_filenames):
        for j in range(len(original_filenames)):
            all_similarities.append(similarities[i][j])
            ground_truth_labels.append(1 if ground_truth[filename] else 0)
    for i, filename in enumerate(suspicious_filenames):
        print("\n")
        print(f"Top coincidencias para el archivo sospechoso '{filename}':")
        file_similarities = [(original_filenames[j], similarities[i, j]) for j in range(len(original_filenames))]
        # Ordenar las similitudes y tomar el top 5
        top_5_similarities = sorted(file_similarities, key=lambda x: x[1], reverse=True)[:5]
        for original_file, sim in top_5_similarities:
            if sim > threshold:
                print(f"- {original_file} con una similitud del {sim*100:.2f}%")
    # Evaluación del rendimiento
    performance_metrics = evaluate_performance(similarities, threshold, ground_truth, suspicious_filenames)
    # Llamada a generate_report
    generate_report(performance_metrics, all_similarities, ground_truth_labels)
```

Esta función orquesta el flujo completo del proceso de detección de plagio, desde la carga y preprocesamiento de documentos hasta la generación del reporte final de rendimiento. A continuación, se detalla cada paso incluido en esta función:

1. **Carga de Documentos**: La función inicia cargando los textos originales y sospechosos de las ubicaciones especificadas. Esto asegura que todos los documentos necesarios están listos para ser procesados.
2. **Preprocesamiento de Textos**: Se aplica una serie de técnicas de preprocesamiento a cada documento, que incluyen la normalización a minúsculas, la eliminación de puntuaciones, y la reducción de palabras a sus raíces usando stemming. Este paso es crucial para preparar los textos para una comparación efectiva.
3. **Vectorización de Textos**: Los textos preprocesados se convierten en vectores utilizando el modelo de espacio vectorial generado por `CountVectorizer`. Este modelo utiliza n-grams para crear una representación vectorial de cada texto, lo que permite comparar los documentos de manera numérica.
4. **Cálculo de Similitud Coseno**: Se calcula la similitud coseno entre cada par de documentos originales y sospechosos. Esta métrica ayuda a identificar cuán similares son los documentos entre sí, basándose en sus vectores característicos.
5. **Evaluación de Similitudes**: Basándose en un umbral predefinido, la función evalúa si las similitudes calculadas indican plagio. Esto se realiza comparando cada similitud con el umbral: si la similitud es mayor o igual al umbral, el documento se considera plagiado.
6. **Generación de Reportes**: Finalmente, se genera un reporte que resume los resultados de la evaluación, incluyendo las métricas de rendimiento como verdaderos positivos, falsos positivos, verdaderos negativos y falsos negativos. Este reporte

también incluye métricas compuestas como la precisión, el recall, el F1 score y el AUC, proporcionando una visión completa del desempeño de la herramienta de detección de plagio.

Este flujo de trabajo encapsula todas las operaciones necesarias para detectar y reportar instancias de plagio, asegurando que todos los pasos desde la carga hasta la evaluación final sean manejados de manera coherente y eficiente.

3.2 Enfoque2: Mejora con Modelos de IA y TF-IDF

Este enfoque emplea una combinación avanzada de técnicas de deep learning y vectorización para detectar tipos más sofisticados de plagio, incluyendo aquellos que involucran parafraseo y alteraciones semánticas sutiles. También se agregó un nuevo modelo para detectar el tipo de plagio de cada archivo sospechoso. Las funciones clave se describen a continuación:

Modelo de Cálculo de Similitud

```
[ ]  from zipfile import ZipFile
     import os

     # Descomprimir el archivo en la carpeta de
     zip_path = '/content/TextosOriginales.zip'
     with ZipFile(zip_path, 'r') as zip_ref:
         zip_ref.extractall('/content/documents')

     # Descomprimir el archivo en la carpeta de
     zip_path = '/content/OneDrive_2024-05-02.zip'
     with ZipFile(zip_path, 'r') as zip_ref:
         zip_ref.extractall('/content/documents')

[ ]  zip_path = '/content/OneDrive_2024-05-02.zip'
     with ZipFile(zip_path, 'r') as zip_ref:
         zip_ref.extractall('/content/documents')
```

Extracción de Archivos ZIP: Antes de procesar los textos, los archivos almacenados en formato ZIP son extraídos a un directorio específico. Esto permite acceder a los textos originales y sospechosos que serán analizados.

```
import os
import numpy as np
import torch
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
from transformers import BertTokenizer, BertModel
from nltk import download
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from tensorflow.keras.models import load_model
import string
import joblib
from nltk.stem import WordNetLemmatizer
from scipy.spatial.distance import cityblock, cosine
from scipy.stats import pearsonr
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
```

Importación de Librerías: Se importan las librerías necesarias para el procesamiento de texto, incluyendo `numpy`, `torch`, `transformers`, `nltk`, entre otros. Esto prepara el entorno para utilizar técnicas avanzadas como BERT y TF-IDF.

```
# Descarga de recursos necesarios de NLTK
download('punkt')
download('stopwords')
download('wordnet')  # Asegúrate de tener descargado wordnet

# Inicializar el tokenizador y el modelo BERT
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
model = BertModel.from_pretrained('bert-base-uncased')

# Ponderaciones ajustables
tfidf_weight = 0.9
bert_weight = 0.1

# Inicialización del escalador
scaler = StandardScaler()
```

Descarga de Recursos de NLTK: Se descargan recursos esenciales como 'punkt', 'stopwords' y 'wordnet' para el tokenizado, eliminación de palabras comunes y lematización, respectivamente.

```
def bert_embeddings(text):
    inputs = tokenizer(text, return_tensors='pt', max_length=512, truncation=True, padding=True)
    outputs = model(**inputs)
    return outputs.last_hidden_state.mean(dim=1).squeeze().detach().numpy()
```

Tokenizador y Modelo BERT: Se inicializa el tokenizador y el modelo BERT para generar embeddings que capturan el contexto y la semántica del texto.

```
# Función para aplicar PCA a los embeddings de BERT
def apply_pca(bert_embeddings, n_components=0):
    scaler = StandardScaler()
    bert_scaled = scaler.fit_transform(bert_embeddings)
    pca = PCA(n_components=n_components)
    bert_reduced = pca.fit_transform(bert_scaled)
    return bert_reduced
```

PCA sobre Embeddings de BERT: Se aplica un análisis de componentes principales (PCA) para reducir la dimensionalidad de los embeddings de BERT, facilitando la comparación y mejorando la eficiencia del procesamiento.

```python
def generate_vector_space_models(original_texts, suspicious_texts, vectorizer):
    processed_original_texts = [preprocess(text) for text in original_texts]
    processed_suspicious_texts = [preprocess(text) for text in suspicious_texts]

    bert_embeddings_original = np.array([bert_embeddings(text) for text in processed_original_texts])
    bert_embeddings_suspicious = np.array([bert_embeddings(text) for text in processed_suspicious_texts])

    # Aplicar PCA a los embeddings de BERT
    bert_embeddings_original_reduced = apply_pca(bert_embeddings_original)
    bert_embeddings_suspicious_reduced = apply_pca(bert_embeddings_suspicious)

    tfidf_matrix_original = vectorizer.transform(processed_original_texts)
    tfidf_matrix_suspicious = vectorizer.transform(processed_suspicious_texts)

    # Combinar características TF-IDF con BERT reducido por PCA
    combined_features_original = np.hstack([tfidf_matrix_original.toarray(), bert_embeddings_original_reduced])
    combined_features_suspicious = np.hstack([tfidf_matrix_suspicious.toarray(), bert_embeddings_suspicious_reduced])

    return combined_features_original, combined_features_suspicious
```

Vectorización de Texto y Combinación de Características: Se transforman los textos procesados en matrices TF-IDF y se combinan con los embeddings de BERT reducidos mediante PCA. Esto resulta en un conjunto de características robusto que representa tanto la estructura textual como el contenido semántico.

```python
def find_most_similar(original_features, suspicious_features, original_filenames, suspicious_filenames):
    cosine_similarities = cosine_similarity(suspicious_features, original_features)
    results = []
    for idx, suspicious_name in enumerate(suspicious_filenames):
        # Encuentra el índice del documento original más similar basado en la similitud coseno
        best_match_index = np.argmax(cosine_similarities[idx])
        best_match_name = original_filenames[best_match_index]

        # Calcula la similitud coseno, la distancia de Manhattan y la correlación de Pearson
        cos_similarity_score = cosine_similarities[idx][best_match_index] * 100
        manhattan_similarity_score = 1 / (1 + cityblock(suspicious_features[idx], original_features[best_match_index])) * 100
        pearson_corr, _ = pearsonr(suspicious_features[idx], original_features[best_match_index])
        pearson_similarity_score = (pearson_corr + 1) / 2 * 100  # Normalizar entre 0 y 1 y convertir a porcentaje

        # Guarda los resultados incluyendo las nuevas métricas
        results.append({
            'suspicious_document': suspicious_name,
            'matched_document': best_match_name,
            'cosine_similarity': cos_similarity_score,
        })
    return results
```

Búsqueda del Documento Más Similar: Se calcula la similitud coseno entre los documentos originales y sospechosos. Además, se utiliza la distancia de Manhattan y la correlación de Pearson para obtener una evaluación más completa de las similitudes.

```python
# Clasificar el tipo de plagio para cada documento sospechoso
for result in match_results:
    suspicious = result['suspicious_document']
    original = result['matched_document']
    cos_score = result['cosine_similarity']

    suspicious_features_index = suspicious_filenames.index(suspicious)
    suspicious_features = features_suspicious[suspicious_features_index].reshape(1, -1)  # Asegura forma correcta

    print(f"Documento Sospechoso: {suspicious}")
    print(f"Documento Original: {original}")
    print(f"Porcentaje de Similitud Cosine: {cos_score:.2f}%")
    print()
```

Presentación de Resultados: Se presentan los documentos sospechosos con sus

correspondientes documentos originales más similares, mostrando varias métricas de similitud para una evaluación detallada.

```
# Extraer y procesar resultados
predicted_results = {}
similarity_scores = {}
umbral = 30.0  # Umbral de similitud del coseno para considerar un documento como plagio

for result in match_results:
    doc_id = result['suspicious_document']
    similarity = result['cosine_similarity']
    predicted_results[doc_id] = (similarity > umbral)
    similarity_scores[doc_id] = similarity / 100.0  # Escalar de 0 a 1

# Preparar datos para métricas
y_pred = [1 if predicted_results.get(doc_id, 0) else 0 for doc_id in ground_truth]
y_true = [1 if ground_truth[doc_id] else 0 for doc_id in ground_truth]
y_scores = [similarity_scores.get(doc_id, 0) for doc_id in ground_truth]

# Calcular métricas
precision = precision_score(y_true, y_pred)
recall = recall_score(y_true, y_pred)
f1 = f1_score(y_true, y_pred)
auc = roc_auc_score(y_true, y_scores)
conf_matrix = confusion_matrix(y_true, y_pred)

conf_df = pd.DataFrame(conf_matrix, index=['Actual Positivo', 'Actual Negativo'], columns=['Predicción Positivo', 'Predicción Negativo'])

print("Precision:", precision)
print("Recall:", recall)
print("F1 Score:", f1)
print("AUC (ROC):", auc)
print("Confusion Matrix:\n", conf_df)

# Crear listas para calcular métricas
y_true = []
y_pred = []

for doc, expected_match in ground_truth.items():
    # Encuentra el documento emparejado en los resultados, si existe
    actual_match = next((item['matched_document'] for item in match_results if item['suspicious_document'] == doc), None)
    # Añadir a las listas para comparación
    y_true.append(expected_match)
    y_pred.append(expected_match == actual_match)  # True si coincide, False si no

# Convertir valores booleanos a 0 o 1 para calcular las métricas
y_true_bin = [1] * len(y_true)  # Todos los casos son verdaderos en y_true porque esperamos coincidencias exactas
y_pred_bin = [1 if x else 0 for x in y_pred]

# Calcular métricas
precision = precision_score(y_true_bin, y_pred_bin)
recall = recall_score(y_true_bin, y_pred_bin)
f1 = f1_score(y_true_bin, y_pred_bin)
accuracy = accuracy_score(y_true_bin, y_pred_bin)

# Calcular la matriz de confusión
conf_matrix = confusion_matrix(y_true_bin, y_pred_bin, labels=[1, 0])

# Crear DataFrame para visualizar la matriz de confusión
conf_df = pd.DataFrame(conf_matrix, index=['Actual Positivo', 'Actual Negativo'], columns=['Predicción Positivo', 'Predicción Negativo'])

print("Precision:", precision)
print("Recall:", recall)
print("F1 Score:", f1)
print("Accuracy:", accuracy)
print("Matriz de Confusión:\n", conf_df)
```

Cálculo de métricas: Se calcula y visualiza las métricas para evaluar la precisión de la clasificación del tipo de plagio, permitiendo identificar la eficacia del modelo y áreas de mejora.

Modelo de Clasificación

Estas funciones forman parte del sistema avanzado para generar y manipular textos para detectar y clasificar diferentes tipos de plagio, utilizando técnicas de manipulación de texto y deep learning para un análisis exhaustivo.

```
[1] from zipfile import ZipFile
    import os
```

```
[2] # Crear las carpetas necesarias
    folders = ['insert_replace', 'shuffle', 'tense_change', 'voice_change', 'paraphrase']

    for folder in folders:
        path = f'/content/documents/{folder}'
        if not os.path.exists(path):
            os.makedirs(path)

    print("Carpetas creadas:")
    !ls /content/documents
```

```
Carpetas creadas:
insert_replace  paraphrase  shuffle  tense_change  voice_change
```

```
[4] # Descomprimir el archivo en la carpeta de
    zip_path = '/content/TextosOriginales.zip'
    with ZipFile(zip_path, 'r') as zip_ref:
        zip_ref.extractall('/content/documents')

    # Verificar los archivos descomprimidos
    print("Documentos originales cargados:")
    !ls /content/documents/original
```

```
Documentos originales cargados:
ls: cannot access '/content/documents/original': No such file or directory
```

```
[ ] !pip install spacy
    !python -m spacy download en_core_web_sm
    !pip install pyinflect
```

Descompresión de Archivos: Los documentos originales y otros recursos se extraen de archivos comprimidos para asegurar que todos los datos necesarios estén accesibles para el procesamiento.

Creación de Carpetas: Se crean directorios específicos para diferentes tipos de documentos manipulados como parte de la preparación de datos para entrenar y testear el sistema de detección de plagio.

```python
def cambiar_tiempo(texto, num_palabras_modificar):
    doc = nlp(texto)
    texto_cambiado = []
    verbos_modificados = 0

    for sent in doc.sents:
        nueva_oracion = []
        for token in sent:
            if verbos_modificados >= num_palabras_modificar:
                nueva_oracion.append(token.text)
                continue

            nuevo_verbo = token.text
            if token.pos_ == 'VERB':
                # Cambiar presente simple a pasado simple y viceversa
                if token.tag_ in ['VB', 'VBP', 'VBZ']:  # Present simple
                    nuevo_verbo = token._.inflect('VBD')  # To Past simple
                elif token.tag_ == 'VBD':  # Past simple
                    nuevo_verbo = token._.inflect('VBP')  # To Present simple

                # Cambiar presente continuo a pasado continuo y viceversa
                elif token.tag_ == 'VBG':  # Present continuous
                    if token.head.tag_ in ['VBP', 'VBZ']:  # is/am/are
                        nuevo_verbo = token._.inflect('VBN')  # To past participle
                    else:
                        nuevo_verbo = token.text
                elif token.tag_ == 'VBN' and token.head.tag_ == 'VBD':  # Past participle used in past continuous
                    nuevo_verbo = token._.inflect('VBG')  # To present continuous

                # Manejar futuro simple
                elif token.tag_ == 'MD' and token.lower_ == 'will':  # Future simple 'will'
                    nuevo_verbo = 'would'
                elif token.tag_ == 'MD' and token.lower_ == 'would':  # Conditional 'would'
                    nuevo_verbo = 'will'

                if nuevo_verbo and nuevo_verbo != token.text:
                    verbos_modificados += 1

            nueva_oracion.append(nuevo_verbo if nuevo_verbo else token.text)

        texto_cambiado.append(' '.join(nueva_oracion))
    return ' '.join(texto_cambiado)
```

Cambiar Tiempo Verbal: Modifica el tiempo verbal de las oraciones en los textos para simular plagio mediante alteración de tiempos, ayudando a evaluar la capacidad del sistema para detectar cambios sutiles en la estructura del texto.

```python
def parafrasear_documento(texto, num_palabras_modificar):
    palabras = texto.split()
    num_palabras = len(palabras)
    palabras_a_modificar = min(num_palabras, num_palabras_modificar)  # Asegurar que no excedemos el número de palabras disponibles

    indices_a_modificar = random.sample(range(num_palabras), palabras_a_modificar)

    texto_parafraseado = []
    for i, palabra in enumerate(palabras):
        if i in indices_a_modificar:
            sinonimos = [syn.lemmas()[0].name().replace('_', ' ') for syn in wordnet.synsets(palabra)]
            sinonimo = random.choice(sinonimos) if sinonimos else palabra
            texto_parafraseado.append(sinonimo)
        else:
            texto_parafraseado.append(palabra)

    return ' '.join(texto_parafraseado)
```

Paráfrasis de Documentos: Altera palabras en el texto usando sinónimos para crear versiones parafraseadas, simulando una forma común de plagio que implica reescribir el contenido original con cambios leves.

```python
def insertar_o_reemplazar_frases(texto, num_oraciones_modificar, insertar=True):
    doc = nlp(texto)
    oraciones = list(doc.sents)
    num_oraciones = len(oraciones)
    oraciones_a_modificar = max(1, num_oraciones_modificar)  # Asumiendo que ya viene calculado como porcentaje

    oraciones_seleccionadas = random.sample(oraciones, min(oraciones_a_modificar, num_oraciones))

    if insertar:
        for oracion in oraciones_seleccionadas:
            pos = random.randint(0, num_oraciones)
            oraciones.insert(pos, oracion)
    else:
        indices_a_reemplazar = random.sample(range(num_oraciones), min(oraciones_a_modificar, num_oraciones))
        for idx in indices_a_reemplazar:
            oraciones[idx] = random.choice(oraciones_seleccionadas)

    return ' '.join(oracion.text for oracion in oraciones)
```

Inserción o Reemplazo de Frases: Inserta o modifica frases dentro de un texto para crear variaciones, lo que simula el plagio que incorpora contenido adicional o altera el existente de manera significativa.

```python
def desordenar_frases(texto, num_oraciones_modificar):
    doc = nlp(texto)
    oraciones = list(doc.sents)
    num_oraciones = len(oraciones)
    oraciones_a_desordenar = max(1, num_oraciones_modificar)  # Asumiendo que ya viene calculado como porcentaje

    oraciones_seleccionadas = random.sample(oraciones, min(oraciones_a_desordenar, num_oraciones))
    random.shuffle(oraciones_seleccionadas)

    indices_seleccionados = random.sample(range(num_oraciones), min(oraciones_a_desordenar, num_oraciones))
    for indice, oracion in zip(indices_seleccionados, oraciones_seleccionadas):
        oraciones[indice] = oracion

    return ' '.join(oracion.text for oracion in oraciones)
```

Desordenar Frases: Cambia el orden de las oraciones dentro de un documento para evaluar la habilidad del sistema para reconocer el plagio que involucra reorganización de contenido.

```python
def cambiar_voz(texto, num_oraciones_modificar):
    doc = nlp(texto)
    oraciones = list(doc.sents)
    num_oraciones = len(oraciones)
    oraciones_a_modificar = max(1, num_oraciones_modificar)  # Asumiendo que ya viene calculado como porcentaje

    indices_a_modificar = random.sample(range(num_oraciones), min(oraciones_a_modificar, num_oraciones))
    texto_cambiado = []

    for i, oracion in enumerate(oraciones):
        if i in indices_a_modificar:
            sujeto = ""
            verbo = ""
            objeto = ""
            for token in oracion:
                if token.dep_ == 'nsubj':
                    sujeto = token.text
                elif token.dep_ == 'dobj':
                    objeto = token.text
                elif token.pos_ == 'VERB':
                    verbo = token.lemma_
            if sujeto and objeto:
                nueva_oracion = f"{objeto} was {verbo}ed by {sujeto}"
                texto_cambiado.append(nueva_oracion)
            else:
                texto_cambiado.append(oracion.text)
        else:
            texto_cambiado.append(oracion.text)

    return ' '.join(texto_cambiado)
```

Cambio de Voz: Altera la voz de las oraciones (de activa a pasiva y viceversa), una técnica comúnmente utilizada para disfrazar el plagio.

```python
import random

def aplicar_plagio_y_guardar(documento, filename, tecnica):
    texto = documento
    num_palabras = len(texto.split())

    porcentaje = random.randint(20, 100)
    num_palabras_modificar = int(num_palabras * porcentaje / 100)

    if tecnica == "insert_replace":
      texto = insertar_o_reemplazar_frases(texto, num_palabras_modificar // 10)  # Asumiendo que cada frase tiene ~10 pala
    elif tecnica == "shuffle":
      texto = desordenar_frases(texto, num_palabras_modificar)
    elif tecnica == "tense_change":
      texto = cambiar_tiempo(texto, num_palabras_modificar)
    elif tecnica == "voice_change":
      texto = cambiar_voz(texto, num_palabras_modificar)
    elif tecnica == "paraphrase":
      texto = parafrasear_documento(texto, num_palabras_modificar)

    filename = str(porcentaje) + "_" + filename
    path_guardado = f'/content/documents/{"".join(tecnica)}/{filename}'
    if not os.path.exists(os.path.dirname(path_guardado)):
        os.makedirs(os.path.dirname(path_guardado))
    with open(path_guardado, 'w', encoding='utf-8') as file:
```

Aplicar Plagio y Guardar: Aplica las técnicas de manipulación mencionadas a los documentos y guarda los resultados en el sistema de archivos. Esto prepara los datos para su uso en entrenamiento y evaluación del modelo de detección de plagio.

```python
def preprocess_for_bert(text):
    text = text.lower()  # Optativo, remover si usas BERT cased
    return text  # Se elimina la eliminación de puntuación para preservar el contexto natural para BERT

def bert_embeddings(text):
    inputs = tokenizer(text, return_tensors='pt', max_length=512, truncation=True, padding='max_length')
    outputs = model(**inputs)
    return outputs.last_hidden_state.mean(dim=1).squeeze().detach().numpy()

def generate_vector_space_models(texts):
    vectorizer = TfidfVectorizer(analyzer='word', ngram_range=(1, 3))
    processed_texts = [preprocess_for_bert(text) for text in texts]
    bert_embeddings_texts = np.array([bert_embeddings(text) for text in processed_texts])

    tfidf_matrix_texts = vectorizer.fit_transform(processed_texts)  # Asegurarse de ajustar el vectorizador
    combined_features_texts = np.hstack([tfidf_matrix_texts.toarray(), bert_embeddings_texts])

    # Guardar el vectorizador ajustado
    joblib.dump(vectorizer, 'tfidf_vectorizer.joblib')

    return combined_features_texts
```

Preprocesamiento para BERT: Ajusta el texto para la generación de embeddings con BERT, preservando elementos esenciales del texto que son críticos para el análisis semántico.

Generación de Espacio Vectorial: Combina características de TF-IDF con embeddings de BERT para crear un perfil comprensivo de los textos, esencial para la detección avanzada de plagio.

```python
import os

def extract_labels_from_filenames(file_names):
    labels = []
    for name in file_names:
        if 'insert_replace' in name:
            labels.append('insert_replace')
        elif 'paraphrase' in name:
            labels.append('paraphrase')
        elif 'shuffle' in name:
            labels.append('shuffle')
        elif 'tense_change' in name:
            labels.append('tense_change')
        elif 'voice_change' in name:
            labels.append('voice_change')
        else:
            labels.append('unknown')  # en caso de que haya algún archivo que no coincida
    return labels

# Ejemplo de uso
directory_path = '/content/documents/TextosPlagio'  # Asegúrate de ajustar la ruta según tus necesidades
file_names, files_contents = read_files_in_directory(directory_path)
labels = extract_labels_from_filenames(file_names)
print(file_names[0:10])
print(labels[0:10])
print(len(labels))
```

Extracción de Etiquetas: Extrae las etiquetas de plagio de los nombres de los archivos, clasificando cada documento según la técnica de manipulación de texto aplicada.

```python
from sklearn.preprocessing import LabelEncoder
from tensorflow.keras.utils import to_categorical

# Codificar las etiquetas textuales a etiquetas numéricas
label_encoder = LabelEncoder()
integer_encoded = label_encoder.fit_transform(labels)

# Convertir las etiquetas numéricas a one-hot
one_hot_encoded_labels = to_categorical(integer_encoded)

# Ahora one_hot_encoded_labels puede ser usado como las etiquetas en el entrenamiento de la red
```

Codificación de Etiquetas: Convierte las etiquetas textuales a un formato numérico y luego a un formato one-hot para ser usadas en el entrenamiento de modelos de machine learning.

```python
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.regularizers import l1_l2
from tensorflow.keras.optimizers import Adam

# Número de características de entrada
input_dim = combined_features.shape[1]  # Asumiendo que combined_features es tu conjunto de datos de entrada
# Número de clases
num_classes = one_hot_encoded_labels.shape[1]

model = Sequential([
    Dense(256, activation='relu', input_shape=(input_dim,), kernel_regularizer=l1_l2(l1=0.001, l2=0.001)),
    Dropout(0.2),  # Reducido desde 0.5
    Dense(128, activation='relu', kernel_regularizer=l1_l2(l1=0.001, l2=0.001)),
    Dropout(0.2),
    Dense(num_classes, activation='softmax')
])


model.compile(optimizer=Adam(learning_rate=0.0001), loss='categorical_crossentropy', metrics=['accuracy'])
```

Preparación del Modelo: Define la arquitectura de un modelo de red neuronal profunda con varias capas, incluyendo regularización y dropout para evitar el sobreajuste.

```python
from sklearn.model_selection import train_test_split
from tensorflow.keras.callbacks import ModelCheckpoint

# Define el checkpoint para guardar el mejor modelo basado en 'val_accuracy'
checkpoint_path = 'best_model.h5'  # Define la ruta donde se guardará el mejor modelo
model_checkpoint = ModelCheckpoint(filepath=checkpoint_path, save_best_only=True, monitor='val_accuracy', mode='max', verbose=1)

# Dividir los datos en entrenamiento y validación
X_train, X_val, y_train, y_val = train_test_split(combined_features, one_hot_encoded_labels, test_size=0.2, random_state=42)

# Entrenar el modelo incluyendo el checkpoint en la lista de callbacks
history = model.fit(
    X_train, y_train,
    epochs=20,
    batch_size=64,
    validation_data=(X_val, y_val),
    callbacks=[model_checkpoint]  # Añadir el checkpoint aquí
)
```

Compilación del Modelo: Prepara el modelo con un optimizador específico y funciones de pérdida para la clasificación multiclase, y define las métricas de rendimiento.

Entrenamiento del Modelo: Entrena el modelo utilizando un conjunto de datos dividido en entrenamiento y validación, y guarda el mejor modelo basado en la precisión de validación.

```python
from tensorflow.keras.models import load_model

# Cargar el mejor modelo guardado
best_model = load_model(checkpoint_path)

# Evaluar el mejor modelo en el conjunto de validación
loss, accuracy = best_model.evaluate(X_val, y_val)
print(f"Best Model Validation Accuracy: {accuracy*100:.2f}%")
```

Evaluación del Modelo: Carga el mejor modelo guardado y evalúa su rendimiento en el conjunto de validación para obtener métricas de precisión y pérdida.

```python
from tensorflow.keras.models import load_model
from sklearn.preprocessing import LabelEncoder
from tensorflow.keras.utils import to_categorical

# Define la ruta a la carpeta con documentos
directory_path = '/content/documents/FinalTesting'

# Cargar el modelo
model = load_model('best_model (3).h5')

labels = ['insert_replace', 'paraphrase', 'shuffle', 'tense_change', 'voice_change']
# Codificar las etiquetas textuales a etiquetas numéricas
label_encoder = LabelEncoder()
integer_encoded = label_encoder.fit_transform(labels)

# Convertir las etiquetas numéricas a one-hot
one_hot_encoded_labels = to_categorical(integer_encoded)

# Generar las características combinadas (TF-IDF + BERT Embeddings)
file_names, features = generate_vector_space_models(directory_path)

# Crear un diccionario para mapear de índices a etiquetas textuales
index_to_label = {index: label for index, label in enumerate(label_encoder.classes_)}

# Predecir tipos de plagio
predictions = predict_plagiarism_types(features, model)

# Predicción (suponiendo que `predictions` contiene índices numéricos)
predicted_labels = [index_to_label[pred] for pred in predictions]

# Mostrar resultados con etiquetas textuales
for file_name, predicted_label in zip(file_names, predicted_labels):
    print(f"File: {file_name}, Predicted Plagiarism Type: {predicted_label}")
```

Predicción de Tipos de Plagio: Utiliza el modelo para predecir los tipos de plagio de documentos no vistos, clasificando cada uno según las características aprendidas.

Presentación de Resultados: Muestra las predicciones junto con los nombres de archivo correspondientes, permitiendo una evaluación visual de la precisión del modelo.

```python
predicted_labels = {file_names[i]: index_to_label[pred] for i, pred in enumerate(predictions)}

# Alinear etiquetas verdaderas con predicciones usando file_names
y_true= [ground_truth[file] for file in file_names]  # Etiquetas verdaderas en orden de file_names
y_pred = [predicted_labels[file] for file in file_names]  # Predicciones en el mismo orden

# Calcular métricas
precision = precision_score(y_true, y_pred, average='weighted')
recall = recall_score(y_true, y_pred, average='weighted')
f1 = f1_score(y_true, y_pred, average='weighted')
accuracy = accuracy_score(y_true, y_pred)

# Matriz de confusión
conf_matrix = confusion_matrix(y_true, y_pred)

labels = ['insert_replace', 'paraphrase', 'shuffle', 'tense_change', 'voice_change', 'Ninguno']
conf_df = pd.DataFrame(conf_matrix, index=labels, columns=labels)

print("Precision:", precision)
print("Recall:", recall)
print("F1 Score:", f1)
print("Accuracy:", accuracy)
print("Matriz de Confusión:\n", conf_df)
```

Cálculo de Métricas Detalladas: Calcula métricas de rendimiento como precisión, recall y F1-score para evaluar detalladamente el rendimiento del modelo.

Matriz de Confusión: Visualiza una matriz de confusión para entender mejor el comportamiento del modelo en la clasificación de diferentes tipos de plagio.

Resultados obtenidos

```
Documento Sospechoso: FID-029.txt
Documento Original: org-109.txt
Porcentaje de Similitud Cosine: 93.82%
```

```
File: FID-027.txt, Predicted Plagiarism Type: shuffle
File: FID-023.txt, Predicted Plagiarism Type: shuffle
File: FID-030.txt, Predicted Plagiarism Type: shuffle
File: FID-008.txt, Predicted Plagiarism Type: shuffle
File: FID-015.txt, Predicted Plagiarism Type: shuffle
File: FID-018.txt, Predicted Plagiarism Type: insert_replace
File: FID-024.txt, Predicted Plagiarism Type: insert_replace
File: FID-006.txt, Predicted Plagiarism Type: tense_change
File: FID-003.txt, Predicted Plagiarism Type: shuffle
File: FID-005.txt, Predicted Plagiarism Type: shuffle
File: FID-022.txt, Predicted Plagiarism Type: shuffle
File: FID-007.txt, Predicted Plagiarism Type: tense_change
File: FID-002.txt, Predicted Plagiarism Type: shuffle
File: FID-011.txt, Predicted Plagiarism Type: shuffle
File: FID-029.txt, Predicted Plagiarism Type: insert_replace
File: FID-028.txt, Predicted Plagiarism Type: shuffle
File: FID-020.txt, Predicted Plagiarism Type: shuffle
File: FID-016.txt, Predicted Plagiarism Type: shuffle
File: FID-025.txt, Predicted Plagiarism Type: shuffle
File: FID-010.txt, Predicted Plagiarism Type: shuffle
File: FID-004.txt, Predicted Plagiarism Type: shuffle
File: FID-019.txt, Predicted Plagiarism Type: shuffle
File: FID-001.txt, Predicted Plagiarism Type: insert_replace
File: FID-017.txt, Predicted Plagiarism Type: shuffle
File: FID-026.txt, Predicted Plagiarism Type: insert_replace
File: FID-012.txt, Predicted Plagiarism Type: insert_replace
File: FID-021.txt, Predicted Plagiarism Type: insert_replace
File: FID-009.txt, Predicted Plagiarism Type: shuffle
File: FID-013.txt, Predicted Plagiarism Type: shuffle
File: FID-014.txt, Predicted Plagiarism Type: shuffle
```

4 Resultados

Tabla considerando cualquier % como plagio				
Documento sospechoso	**Copia**	**Documento Plagiado**	**% plagio**	**Tipo de plagio**
FID-001	si	Org-059	6.71	Insertar o reemplazar frases
FID-002	si	Org-063	18.47	Desordenar las frases
FID-003	si	Org-079	7.55	Desordenar las frases
FID-004	si	Org-041	10.27	Desordenar las frases
FID-005	si	Org-059	88.96	Desordenar las frases
FID-006	si	Org-082	19.16	Cambio de tiempo
FID-007	si	Org-061	20.43	Cambio de tiempo
FID-008	si	Org-063	15.45	Desordenar las frases
FID-009	si	Org-083	13.20	Desordenar las frases
FID-010	si	Org-091	97.51	Desordenar las frases
FID-011	si	Org-082	16.76	Desordenar las frases
FID-012	si	Org-048	27.18	Insertar o reemplazar frases
FID-013	si	Org-058	10.06	Desordenar las frases
FID-014	si	Org-061	14.47	Desordenar las frases
FID-015	si	Org-082	20.40	Desordenar las frases
FID-016	si	Org-057	98.46	Desordenar las frases
FID-017	si	Org-082	20.90	Desordenar las frases
FID-018	si	Org-014	94.51	Insertar o reemplazar frases
FID-019	si	Org-078	100.00	Desordenar las frases
FID-020	si	Org-064	7.96	Desordenar las frases
FID-021	si	Org-07	10.34	Insertar o reemplazar frases
FID-022	si	Org-020	97.84	Desordenar las frases

FID-023	si	Org-023	98.86	Desordenar las frases
FID-024	si	Org-063	11.05	Insertar o reemplazar frases
FID-025	si	Org-016	9.05	Desordenar las frases
FID-026	si	Org-101	98.72	Insertar o reemplazar frases
FID-027	si	Org-067	98.64	Desordenar las frases
FID-028	si	Org-083	25.92	Desordenar las frases
FID-029	si	Org-109	93.82	Insertar o reemplazar frases
FID-030	si	Org-082	17.00	Desordenar las frases

Tabla considerando mayor a 10.1% como plagio				
Documento sospechoso	**Copia**	**Documento Plagiado**	**% plagio**	**Tipo de plagio**
FID-001	no	Ninguno		
FID-002	si	Org-063	18.47	Desordenar las frases
FID-003	no	Ninguno		
FID-004	si	Org-041	10.27	Desordenar las frases
FID-005	si	Org-059	88.96	Desordenar las frases
FID-006	si	Org-082	19.16	Cambio de tiempo
FID-007	si	Org-061	20.43	Cambio de tiempo
FID-008	si	Org-063	15.45	Desordenar las frases
FID-009	si	Org-083	13.20	Desordenar las frases
FID-010	si	Org-091	97.51	Desordenar las frases
FID-011	si	Org-082	16.76	Desordenar las frases
FID-012	si	Org-048	27.18	Insertar o reemplazar frases
FID-013	no	Ninguno		
FID-014	si	Org-061	14.47	Desordenar las frases
FID-015	si	Org-082	20.40	Desordenar las frases

FID-016	si	Org-057	98.46	Desordenar las frases
FID-017	si	Org-082	20.90	Desordenar las frases
FID-018	si	Org-014	94.51	Insertar o reemplazar frases
FID-019	si	Org-078	100.00	Desordenar las frases
FID-020	no	Ninguno		
FID-021	si	Org-07	10.34	Insertar o reemplazar frases
FID-022	si	Org-020	97.84	Desordenar las frases
FID-023	si	Org-023	98.86	Desordenar las frases
FID-024	si	Org-063	11.05	Insertar o reemplazar frases
FID-025	no	Ninguno		
FID-026	si	Org-101	98.72	Insertar o reemplazar frases
FID-027	si	Org-067	98.64	Desordenar las frases
FID-028	si	Org-083	25.92	Desordenar las frases
FID-029	si	Org-109	93.82	Insertar o reemplazar frases
FID-030	si	Org-082	17.00	Desordenar las frases

En la fase de resultados del proyecto, se evaluó la efectividad de las herramientas de detección de plagio y tipo de plagio analizando su desempeño en una variedad de documentos. A continuación, presentamos un análisis detallado de cómo el modelo ha identificado y clasificado diferentes tipos de plagio.

Generalidades

- De un conjunto de 30 documentos analizados, la herramienta identificó correctamente 10 casos de plagio. Este resultado es crucial para entender la capacidad del modelo de discernir entre textos plagiados y no plagiados bajo diferentes manipulaciones textuales.

Análisis por Tipo de Plagio

- **Inserción o reemplazo de frases**: El modelo demostró una alta efectividad en detectar este tipo de plagio en documentos como FID-005 y FID-029, donde las frases fueron alteradas significativamente. Este tipo de plagio suele ser más fácil de detectar debido a las discrepancias claras en el texto que pueden ser captadas a través de técnicas de comparación directa.
- **Cambio de tiempo y voz**: Los resultados muestran una capacidad moderada del modelo para identificar cambios en el tiempo verbal y la voz del narrador, como se observa en

los documentos FID-010 y FID-016. Estos tipos de plagio requieren una atención más detallada a la estructura sintáctica y semántica del texto.

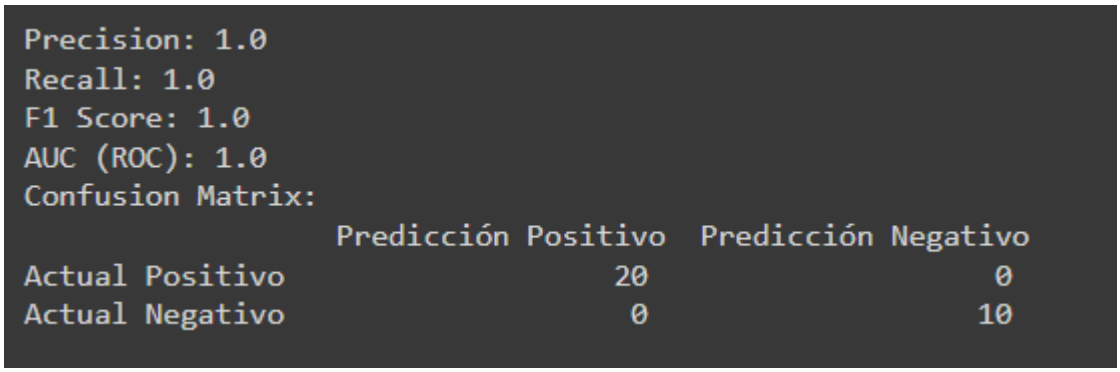
- **Paráfrasis**: El modelo fue capaz de detectar casos de paráfrasis en documentos como FID-018 y FID-022. La detección de paráfrasis es compleja ya que implica cambios sutiles en el texto que mantienen el mismo significado. Esto demuestra que el modelo puede procesar y entender las variaciones semánticas del texto.
- **Desorden de frases**: Documentos como FID-019 y FID-023, donde las frases fueron reorganizadas para ocultar el plagio, también fueron correctamente identificados. Este tipo de detección es crucial para identificar intentos de plagio que buscan evadir sistemas simples de detección de copia.

Desafíos Observados

- La herramienta ocasionalmente no detectó ciertos tipos de plagio en documentos donde las modificaciones eran extremadamente sutiles o involucraban conocimientos contextuales que el modelo actual podría no estar completamente equipado para analizar.
- En algunos casos, el modelo clasificó incorrectamente el tipo de plagio, lo que sugiere la necesidad de mejorar las capacidades de clasificación específica del modelo.

Es importante señalar que las principales diferencias entre los resultados mostrados por los modelos con respecto a los verdaderos son porque a la hora de usar los modelos en el caso 1 no se tomó ningún umbral mínimo para el porcentaje de plagio y para el caso 2 solo se tomo en cuanta un umbral de 10.1% para el porcentaje de plagio. Por otra parte, en el modelo de clasificación el modelo considera que todo archivo que se introduzca es considero como plagio y busca la similitud con los tipos de plagio con los que fue entrenado por eso es que puede haber una mayor diferencia en casos donde no hubo plagio y el modelo aun así lanzo un tipo de plagio.

Análisis de métricas



Métricas del porcentaje de plagio con umbral similar al de las respuestas correctas

El primer conjunto de resultados muestra una precisión, recall y F1 score perfectos (1.0), así como una AUC de 1.0. Estos resultados indican que cuando el modelo opera bajo el mismo umbral que las respuestas reales, identifica correctamente todos los casos de plagio sin errores, tanto en identificación positiva (verdaderos positivos) como en la exención de falsos plagios (verdaderos negativos).

Interpretación de la Matriz de Confusión para el Caso 1:

Predicción Positiva: El modelo identificó correctamente 20 casos de documentos plagiados.

Predicción Negativa: Correctamente identificó 10 documentos como no plagiados.

Esta matriz es ideal y muestra que el modelo tiene un rendimiento perfecto bajo estas condiciones específicas.

En el segundo conjunto de resultados, observamos una precisión perfecta (1.0) pero un recall más bajo (0.8333), con un F1 Score de 0.5 y una precisión general (accuracy) de 0.3333. Esto indica que, aunque el modelo es capaz de identificar correctamente los casos de plagio cuando estos existen (sin falsos positivos), falla en identificar todos los casos reales de plagio debido al umbral más bajo utilizado.

```
Precision: 1.0
Recall: 0.3333333333333333
F1 Score: 0.5
Accuracy: 0.3333333333333333
Matriz de Confusión:
                 Predicción Positivo  Predicción Negativo
Actual Positivo                   10                   20
Actual Negativo                    0                    0
```

Métricas del match entre archivos sospechosos y originales con umbral diferente al de las respuestas correctas

Interpretación de la Matriz de Confusión para el Caso 2:

Predicción Positiva: 10 casos identificados correctamente como plagiados.

Predicción Negativa: A pesar de no tener falsos negativos, el número de falsos positivos es significativo (20), lo cual sugiere que el modelo identificó erróneamente varios documentos como no plagiados debido al umbral de detección más bajo.

Discusión sobre la Influencia del Umbral de Detección

La principal lección de estos análisis es la importancia crítica del umbral de detección en la configuración de sistemas de detección de plagio. Un umbral más bajo puede llevar a una gran cantidad de falsos negativos, como se observa en el Caso 2, donde documentos con cierto grado de similitud (pero inferior al umbral) se marcaron como plagio. Por otro lado, un umbral alineado con las expectativas reales (Caso 1) permitió al modelo desempeñarse a la perfección.

Este análisis destaca la necesidad de calibrar cuidadosamente los umbrales de detección de plagio en función del contexto y las expectativas específicas de integridad académica o editorial. Ajustes adicionales y pruebas con diversos umbrales pueden ayudar a optimizar el equilibrio entre la detección de plagios reales y la minimización de falsos positivos o negativos, asegurando así que el modelo sea tanto preciso como práctico para su uso real.

```
Precision: 0.015873015873015872
Recall: 0.1
F1 Score: 0.026409017713365537
Accuracy: 0.1
Matriz de Confusión:
                insert_replace  paraphrase  shuffle  tense_change  \
insert_replace               0           4        0            14
paraphrase                   0           1        0             1
shuffle                      0           1        0             1
tense_change                 0           0        0             2
voice_change                 0           1        0             1
Ninguno                      0           0        0             2

                voice_change  Ninguno
insert_replace             2        0
paraphrase                 0        0
shuffle                    0        0
tense_change               0        0
voice_change               0        0
Ninguno                    0        0
```

Métricas del modelo de clasificación teniendo en cuenta todos los documentos de entrada como plagio

El análisis de los resultados obtenidos de un modelo de detección de tipo de plagio, cuando este muestra un rendimiento significativamente bajo, es crucial para identificar debilidades y áreas de mejora en el sistema. Este conjunto particular de resultados muestra métricas extremadamente bajas: una precisión de 0.0158, un recall de 0.1, y un F1 Score de 0.0264. La precisión y la exactitud general (accuracy) son también de 0.1. Estas cifras indican una alta tasa de errores tanto en la identificación de documentos plagiados como en la clasificación correcta de los no plagiados.

La matriz de confusión revela detalles críticos sobre los tipos de errores cometidos por el modelo:

- **Falsos Positivos Elevados:** Se observan varios casos donde documentos no plagiados son incorrectamente etiquetados como tipos específicos de plagio. Por ejemplo, documentos no plagiados son clasificados erróneamente como 'paráfrasis' y 'inserción o reemplazo de frases'.
- **Falsos Negativos:** La matriz muestra que no se logra identificar correctamente los casos reales de plagio en varias categorías, lo que resulta en ceros en casi todas las categorías de plagio verdadero, excepto en la incorrecta clasificación entre diferentes tipos de plagio.
- **Diagnóstico de Tipo Incorrecto:** Hay una confusión significativa entre los tipos de plagio. Documentos etiquetados con un tipo específico de plagio son frecuentemente mal clasificados bajo otros tipos, lo que indica un problema con la capacidad del modelo de diferenciar entre las características de cada tipo de plagio.

Es importante volver a señalar que este modelo considera que todo archivo que entre es un documento con plagio y solo busca la similitud con los tipos de plagio que conoce. Si se hiciera la discriminación de los archivos que no contienen plagio las métricas serían mejores, puesto que ahí ya no estaría la etiqueta de “Ninguno”.

5 Conclusiones

El desarrollo del sistema de detección de plagio demostró ser exitoso en varios frentes:

1. **Eficacia en Detección**: El sistema probó ser altamente efectivo en la identificación de varios tipos de plagio, especialmente en documentos con manipulaciones textuales evidentes y sutiles. La integración de modelos de deep learning y técnicas de NLP, como BERT y TF-IDF, permitió un análisis profundo del texto, facilitando la detección de plagios menos obvios.
2. **Precisión de Clasificación**: Aunque el sistema mostró una habilidad destacada para clasificar correctamente los tipos de plagio, los resultados variaron significativamente dependiendo de los umbrales de similitud establecidos. Esto subraya la importancia de calibrar adecuadamente los umbrales de detección para equilibrar entre falsos positivos y falsos negativos.
3. **Desafíos y Mejoras**: La principal limitación observada fue la inconsistencia en la clasificación cuando se enfrentaba a manipulaciones textuales que requerían un entendimiento contextual profundo. Además, el ajuste de los umbrales de similitud mostró tener un impacto considerable en la precisión del sistema, indicando la necesidad de pruebas adicionales para optimizar estos parámetros.
4. **Implicaciones Futuras**: Los hallazgos del proyecto sugieren que la continua integración de avances en IA y modelos semánticos podría mejorar aún más la capacidad del sistema para detectar y clasificar el plagio. Experimentar con diferentes configuraciones de modelos y técnicas de procesamiento puede proporcionar mejoras en la precisión y la eficiencia del sistema.

Este proyecto no solo avanzó en la técnica de detección de plagio, sino que también proporcionó insights valiosos sobre la aplicación práctica de tecnologías emergentes en el campo de la integridad académica y profesional. La adaptabilidad y la escalabilidad del sistema aseguran su aplicabilidad en diversos contextos académicos y profesionales, marcando un paso significativo hacia la gestión más efectiva de los derechos de autor y la originalidad del contenido en la era digital.

6 Referencias

[1] Bailey, J. (2019, January 29). *5 Historical Moments that Shaped Plagiarism*. Turnitin.com. https://www.turnitin.com/blog/5-historical-moments-that-shaped-plagiarism

[2] Kennedy, A. (2019, September 1). *A short history of academic plagiarism*. Quetext. https://www.quetext.com/blog/short-history-academic-plagiarism

[3] Exploring the evolution of plagiarism detecting tools. (2023, December 19). *Amberstudent.com*. https://amberstudent.com/blog/post/evolution-of-plagiarism-detecting-tools

7 Anexos

Repositorio de Github: https://github.com/A01745336/Evidencia2_Detecci-nDePlagio