



Tecnológico de Monterrey

Instituto Tecnológico y de Estudios Superiores de Monterrey

Campus Estado de México

Desarrollo de Aplicaciones Avanzadas de ciencias computacionales

Evidencia 1 - Fase 2 - Parte B Implementación usando NLP

Paola Danae López Pérez

A01745689

Diego Alejandro Balderas Tlahuitzo

A01745336

Omar Rodrigo Talavera Becerra

A01752221

Grupo: 201

Fecha de entrega: 21 de abril de 2024

Resumen

El presente proyecto se enfoca en el desarrollo e implementación de un programa avanzado de detección de plagio, diseñado para identificar y evaluar la similitud entre documentos de texto utilizando técnicas de Procesamiento de Lenguaje Natural (NLP) e Inteligencia Artificial (IA). El objetivo principal es proporcionar una herramienta eficiente y precisa que permita detectar la presencia de plagio en documentos y calcular el grado de similitud con respecto a una base de datos de archivos originales.

La metodología empleada se basa en la aplicación de algoritmos de NLP para analizar el contenido semántico, extrayendo características relevantes y comparándolas con los textos originales disponibles en la base de datos. Se utilizan técnicas avanzadas de comparación de textos para identificar patrones y similitudes. El programa desarrollado en Python ha sido diseñado para manejar archivos en formato de texto (.txt). Se han implementado medidas de optimización para garantizar la eficiencia y escalabilidad del sistema, permitiendo su aplicación en entornos con grandes volúmenes de datos.

La validación del programa se ha realizado mediante la comparación con conjuntos de datos de prueba que contienen una variedad de documentos con diferentes niveles de similitud. Los resultados obtenidos han demostrado una alta precisión en la detección de plagio y en la determinación del grado de similitud, validando así la efectividad y fiabilidad del sistema implementado.

Introducción

La detección de plagio ha adquirido una relevancia ineludible en contextos académicos y profesionales, dada la creciente disponibilidad de recursos digitales y la facilidad de acceso a información en línea. En este contexto, el desarrollo e implementación de programas de detección de plagio constituye una medida fundamental para preservar la integridad académica y la originalidad en la producción intelectual. En particular, el uso de técnicas de Procesamiento del Lenguaje Natural (NLP, por sus siglas en inglés) en conjunción con la programación en Python ha emergido como una herramienta prometedora para abordar esta problemática. Este enfoque permite el análisis automático y sistemático de similitudes textuales entre documentos, facilitando la identificación de fragmentos copiados o parafraseados. En este sentido, este proyecto se propone implementar un programa capaz de calcular el porcentaje de similitud de un documento con textos originales para detectar un posible plagio total o parcial.

Objetivos

En función de responder al problema planteado, nuestros objetivos de implementación serán los siguientes:

1. Desarrollar un modelo de NLP robusto: El primer objetivo es construir un modelo de NLP sólido que pueda comprender el contenido semántico y contextual de los documentos de texto. Esto implica entrenar el modelo que pueda identificar patrones y características distintivas.
2. Validar la precisión del sistema: Antes de implementar el programa en un entorno de producción, es fundamental realizar una validación exhaustiva para evaluar su precisión y fiabilidad. Esto implica utilizar conjuntos de datos de prueba que contengan documentos con diferentes niveles de similitud y verificar si el programa puede detectar correctamente el plagio y calcular con precisión el porcentaje de similitud.
3. Crear una interfaz de usuario intuitiva: Por último, pero no menos importante, el programa debe contar con una interfaz de usuario amigable que permita a los usuarios cargar sus documentos, ejecutar el análisis de plagio y visualizar los resultados de manera clara y comprensible.

Metodología

Para tener éxito en la conclusión del objetivo se optó por combinar las metodologías ÁGIL del desarrollo de software junto a un enfoque iterativo incremental. Este método involucra una investigación inicial, aprendizaje experimental y pruebas continuas para garantizar el correcto funcionamiento del código y la entrega del producto en tiempo. A continuación se enlistan los pasos que se llevaron a cabo para la realización del proyecto:

Fase de Investigación:

Se realizó una investigación acerca de los requisitos a cubrir y las necesidades del cliente, estableciendo objetivos claros a alcanzar.

Experimentación:

Se realizaron pequeños proyectos para poner a prueba las nuevas tecnologías, proporcionando la oportunidad de familiarizarnos con nuevos sistemas y poder experimentar con distintos enfoques.

Recursos de testeo:

Se desarrollaron distintos artículos, usando algunas de las metodologías de plagio más comunes (Resumen resultado de insertar o reemplazar frases que pertenecen a un resumen original en otro, también original, resumen resultado de desordenar las frases de un resumen original, resumen resultado de parafrasear un resumen original, incluyendo el uso de sinónimos, antónimos, negación, doble negación, etc.), con la finalidad de probar el producto final.

Desarrollo:

El desarrollo del código se realizó a la par del testing, asegurándose de que cada segmento que se agrega funcionaba correctamente. Para finalizar con la implementación de todo el código en conjunto y poder afinar detalles.

Implementación de código

Para mejorar procesamiento, utilización y presentación del código dividido en 4 pruebas y el código principal:

test_01_reading.py:

El siguiente código define y ejecuta pruebas unitarias para verificar el estado en el que se encuentran los directorios que se analizarán en la resolución del reto. Comparando los directorios con los textos a analizar y los originales

```

1  import unittest
2  from plagiarism_detection import read_files_in_directory
3
4  class TestFileReading(unittest.TestCase):
5      def test_empty_directory(self):
6          filenames, contents = read_files_in_directory('path/to/empty/directory')
7          self.assertEqual(len(filenames), 0)
8          self.assertEqual(len(contents), 0)
9
10         def test_nonexistent_directory(self):
11             with self.assertRaises(Exception):
12                 read_files_in_directory('path/to/nonexistent/directory')
13
14         def test_reading_contents(self):
15             filenames, contents = read_files_in_directory('path/to/test/directory')
16             self.assertGreater(len(filenames), 0)
17             self.assertEqual(len(filenames), len(contents))
18
19     if __name__ == '__main__':
20         unittest.main()

```

test_02_preprocessing.py:

Este segmento se encarga de las pruebas unitarias para verificar el procesamiento que se le da al texto, como es la eliminación de los signos de puntuación, la aplicación del stemming y un filtro para entradas que no son strings

```

1  import unittest
2  from plagiarism_detection import preprocess
3
4  class TestPreprocessing(unittest.TestCase):
5      def test_lowercase(self):
6          self.assertEqual(preprocess("Test"), "test")
7
8      def test_remove_punctuation(self):
9          self.assertEqual(preprocess("Hello, world!"), "hello world")
10
11         def test_stemming(self):
12             self.assertIn("run", preprocess("running"))
13
14         def test_non_string_input(self):
15             # Verifica que se maneje correctamente una entrada que no es un string
16             with self.assertRaises(AttributeError):
17                 preprocess(123) # Pasando un número en lugar de un string
18
19     if __name__ == '__main__':
20         unittest.main()

```

test_03_vectorization.py:

Este segmento realiza las pruebas unitarias para generar el modelo del vector. Esto ara garantizar el correcto funcionamiento de la función en el contexto de plagio.

```

1 import unittest
2 from plagiarism_detection import generate_vector_space_models
3
4 class TestVectorization(unittest.TestCase):
5     def test_vector_output(self):
6         texts = ["hello world", "hello"]
7         vectors, feature_names = generate_vector_space_models(texts, texts)
8         self.assertEqual(len(feature_names), 2)
9         self.assertEqual(vectors.shape[0], 2)
10        self.assertEqual(vectors.shape[1], 2)
11
12        def test_empty_input(self):
13            # Verifica que se maneje correctamente la entrada vacía
14            with self.assertRaises(ValueError):
15                generate_vector_space_models([], []) # Entradas vacías
16
17
18
19 if __name__ == '__main__':
20     unittest.main()

```

test_04_evaluation.py:

Este segmento asegura el correcto cálculo de AUC y el manejo adecuado de casos en los que las longitudes de las listas no coinciden.

```

1 import unittest
2 from plagiarism_detection import evaluate_performance, generate_report
3 from sklearn.metrics import roc_curve, auc
4
5 class TestEvaluation(unittest.TestCase):
6     def test_performance_metrics(self):
7         suspicious_filenames = ['doc1.txt', 'doc2.txt', 'doc3.txt']
8         similarities = [
9             [0.5, 0.2], # Similitudes de doc1.txt con cada documento original
10            [0.3, 0.6], # Similitudes de doc2.txt con cada documento original
11            [0.7, 0.1] # Similitudes de doc3.txt con cada documento original
12        ]
13        ground_truth = {'doc1.txt': True, 'doc2.txt': False, 'doc3.txt': True}
14        results = evaluate_performance(similarities, 0.3, ground_truth, suspicious_filenames)
15        self.assertEqual(results['TP'], 2)
16        self.assertEqual(results['TN'], 0)
17        self.assertEqual(results['FP'], 1)
18        self.assertEqual(results['FN'], 0)
19
20    def test_auc_calculation(self):
21        # AUC should be correctly calculated
22        similarities = [0, 0.5, 1]
23        ground_truth = [0, 1, 1]
24        fpr, tpr, thresholds = roc_curve(ground_truth, similarities)
25        auc_value = auc(fpr, tpr)
26        self.assertAlmostEqual(auc_value, 1)
27
28    def test_mismatched_lengths(self):
29        # Verifica que se maneje correctamente cuando las longitudes no coinciden
30        suspicious_filenames = ['doc1.txt', 'doc2.txt']
31        similarities = [[0.5, 0.2]] # Solo un elemento
32        ground_truth = { 'doc1.txt': True, 'doc2.txt': False } # Dos elementos
33        with self.assertRaises(IndexError):
34            evaluate_performance(similarities, 0.5, ground_truth, suspicious_filenames)
35
36 if __name__ == '__main__':
37     unittest.main()

```

plagiarism_detection_paralel.py:

Este código lleva a cabo un proceso de detección de similitud entre un conjunto de documentos originales y un conjunto de documentos sospechosos de plagio. Vamos a describir cómo funciona y su relación con los códigos anteriores:

1. Importaciones de módulos y recursos necesarios:

- Se importan diversos módulos necesarios para el procesamiento de texto, el cálculo de similitud, el preprocesamiento de archivos y la generación de reportes.
- Se descargan recursos adicionales de NLTK, como modelos de tokenización y stopwords, para el preprocesamiento del texto.

```
1  import os
2  import string
3  import time
4  import numpy as np
5  from nltk import download
6  from nltk.stem import PorterStemmer
7  from nltk.tokenize import word_tokenize
8  from nltk.corpus import stopwords
9  from sklearn.feature_extraction.text import CountVectorizer
10 from sklearn.metrics.pairwise import cosine_similarity
11 from sklearn.metrics import roc_curve, auc
12 from concurrent.futures import ProcessPoolExecutor
13
14 # Descarga de recursos necesarios de NLTK
15 inicio = time.time()
16 download('punkt')
17 download('stopwords')
18
```

2. Funciones de preprocesamiento y generación de modelos de espacio vectorial:

- Se define la función *preprocess* para normalizar, eliminar stopwords y realizar stemming en el texto.

```
42
43  def preprocess(text):
44      """ Procesa el texto aplicando normalización, eliminación de stopwords y stemming. """
45      text = text.lower()
46      text = text.translate(str.maketrans('', '', string.punctuation))
47      tokens = word_tokenize(text)
48      stop_words = set(stopwords.words('english'))
49      stemmer = PorterStemmer()
50      tokens = [stemmer.stem(token) for token in tokens if token not in stop_words]
51      return ' '.join(tokens)
52
```

- Se define la función *generate_vector_space_models* para crear modelos de espacio vectorial común a partir de los textos originales y sospechosos.

```

4  def generate_vector_space_models(original_texts, suspicious_texts):
5      """ Genera modelos de espacio vectorial para textos originales y sospechosos. """
6      vectorizer = CountVectorizer(analyzer='word', ngram_range=(1, 3))
7      # Combina todos los textos para crear un espacio vectorial común
8      combined_texts = original_texts + suspicious_texts
9      vectorizer.fit(combined_texts) # Ajusta el vectorizador a todos los textos
10     original_vectors = vectorizer.transform(original_texts) # Transforma los textos originales
11     suspicious_vectors = vectorizer.transform(suspicious_texts) # Transforma los textos sospechosos
12     return original_vectors, suspicious_vectors, vectorizer.get_feature_names_out()
13

```

3. Funciones de evaluación y generación de reporte:

- Se define la función *evaluate_performance* para evaluar el rendimiento de la detección de plagio utilizando métricas como verdaderos positivos, falsos positivos, etc.

```

65  def evaluate_performance(similarities, threshold, ground_truth):
66      """
67      Evalúa el rendimiento de la herramienta de detección de plagio.
68      similarities: matriz de similitudes entre documentos sospechosos y originales.
69      threshold: el umbral de similitud para considerar un documento como plagiado.
70      ground_truth: dict con clave = nombre del documento sospechoso y valor = bool indicando si es plagiado.
71
72      Retorna un dict con las métricas TP, FP, TN, FN.
73      """
74      TP = FP = TN = FN = 0
75      for i, file_path in enumerate(suspicious_filenames):
76          # Extrae el nombre del archivo de la ruta
77          susp_filename = os.path.basename(file_path)
78          is_plagiarized = ground_truth.get(susp_filename)
79          # Considera el documento plagiado si alguna similitud supera el umbral.
80          detected_as_plagiarized = any(sim > threshold for sim in similarities[i])
81
82          if is_plagiarized and detected_as_plagiarized:
83              TP += 1
84          elif not is_plagiarized and detected_as_plagiarized:
85              FP += 1
86          elif is_plagiarized and not detected_as_plagiarized:
87              FN += 1
88          elif not is_plagiarized and not detected_as_plagiarized:
89              TN += 1
90
91      return {"TP": TP, "FP": FP, "TN": TN, "FN": FN}

```

- Se define la función *generate_report* para generar un informe detallado basado en las métricas de rendimiento calculadas.


```

94  ✓ def generate_report(performance_metrics, similarities, ground_truth_labels):
95      """
96      Genera un informe de rendimiento basado en las métricas dadas.
97      performance_metrics: dict con TP, FP, TN, FN.
98      similarities: lista de valores de similitud entre documentos.
99      ground_truth_labels: lista de etiquetas de verdad fundamental (0 para no plagiado, 1 para plagiado).
100     """
101     TP = performance_metrics["TP"]
102     FP = performance_metrics["FP"]
103     TN = performance_metrics["TN"]
104     FN = performance_metrics["FN"]
105     precision = TP / (TP + FP) if TP + FP > 0 else 0
106     recall = TP / (TP + FN) if TP + FN > 0 else 0
107     accuracy = (TP + TN) / (TP + FP + TN + FN) if TP + FP + TN + FN > 0 else 0
108     f1_score = 2 * (precision * recall) / (precision + recall) if precision + recall > 0 else 0
109
110     # Calcular AUC
111     fpr, tpr, thresholds = roc_curve(ground_truth_labels, similarities)
112     roc_auc = auc(fpr, tpr)
113
114     report = (
115         f"True Positives: {TP}\n"
116         f"False Positives: {FP}\n"
117         f"True Negatives: {TN}\n"
118         f"False Negatives: {FN}\n"
119         f"Precision: {precision:.2f}\n"
120         f"Recall: {recall:.2f}\n"
121         f"Accuracy: {accuracy:.2f}\n"
122         f"F1 Score: {f1_score:.2f}\n"
123         f"AUC (ROC): {roc_auc:.2f}\n"
124     )
125     print(report)

```

4. Procesamiento paralelo de archivos:

- Se define la función *read_and_preprocess_file* para leer y preprocesar un solo archivo.

```

20  ✓ def read_and_preprocess_file(file_path):
21      """Lee y preprocesa un solo archivo."""
22      try:
23          with open(file_path, 'r', encoding='utf-8') as file:
24              content = file.read()
25      except UnicodeDecodeError:
26          with open(file_path, 'r', encoding='windows-1252') as file:
27              content = file.read()
28      # Preprocesar el contenido del archivo
29      processed_content = preprocess(content)
30      return processed_content

```

- Se define la función *parallel_process_files* para procesar en paralelo todos los archivos de texto en un directorio utilizando múltiples procesadores.

```

33  ✓ def parallel_process_files(directory):
34      """Procesa en paralelo todos los archivos de texto en un directorio."""
35      file_paths = [os.path.join(directory, f) for f in os.listdir(directory) if f.endswith('.txt')]
36      processed_texts = []
37      # Ejecuta la lectura y el preprocesamiento en paralelo
38      with ProcessPoolExecutor(max_workers=6) as executor:
39          processed_texts = list(executor.map(read_and_preprocess_file, file_paths, chunksize=10))
40      return file_paths, processed_texts
41

```

5. Flujo principal del script:

- Se definen los directorios de los documentos originales y sospechosos.
- Se paraleliza el procesamiento de los archivos originales y sospechosos utilizando la función *parallel_process_files*.
- Se generan los modelos de espacio vectorial común utilizando la función *generate_vector_space_models*.
- Se calcula la similitud de coseno entre los textos sospechosos y originales.
- Se evalúa el rendimiento del sistema de detección de plagio utilizando la función *evaluate_performance*.
- Se genera un reporte detallado utilizando la función *generate_report*.
- Se imprimen las top coincidencias para cada archivo sospechoso.

6. Medición de tiempo de ejecución:

- Se mide el tiempo de ejecución del proceso.

Resultados

```

Top coincidencias para el archivo sospechoso 'FID-01.txt':
- org-076.txt con una similitud del 64.11%

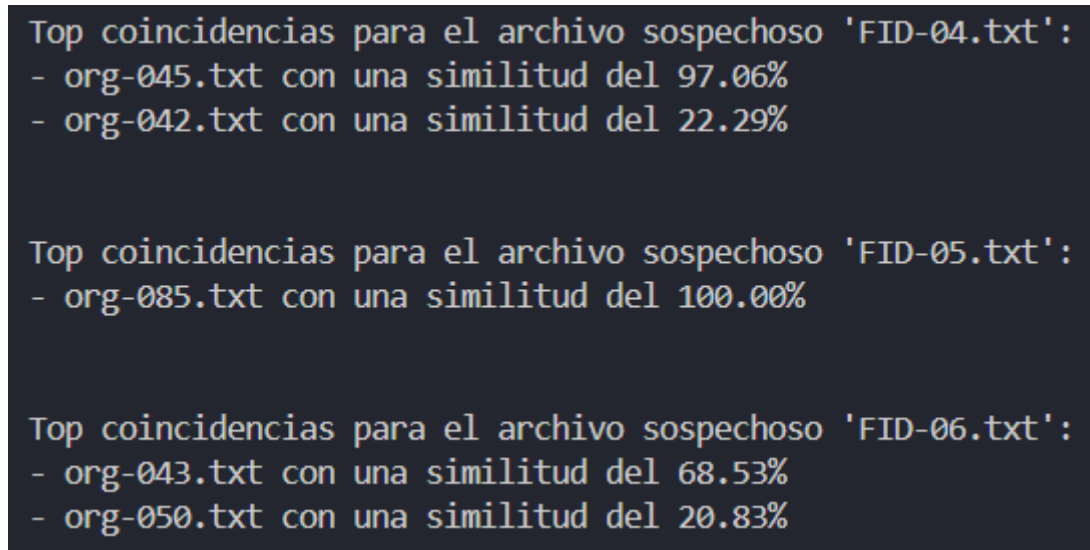
Top coincidencias para el archivo sospechoso 'FID-02.txt':
- org-104.txt con una similitud del 70.54%
- org-082.txt con una similitud del 29.01%

Top coincidencias para el archivo sospechoso 'FID-03.txt':
- org-016.txt con una similitud del 95.59%

```

Imagen . Resultados de la evaluación de los primeros tres archivos

Al observar la imagen podemos darnos cuenta de que al procesar el documento en texto plano, se compara exitosamente con los textos originales almacenados por ahora de forma local. Observamos que el programa nos arroja el nombre del documento analizado y una lista de los archivos con los que encontró coincidencias así como el porcentaje de similitud asociado.



```
Top coincidencias para el archivo sospechoso 'FID-04.txt':  
- org-045.txt con una similitud del 97.06%  
- org-042.txt con una similitud del 22.29%  
  
Top coincidencias para el archivo sospechoso 'FID-05.txt':  
- org-085.txt con una similitud del 100.00%  
  
Top coincidencias para el archivo sospechoso 'FID-06.txt':  
- org-043.txt con una similitud del 68.53%  
- org-050.txt con una similitud del 20.83%
```

Imagen . Resultados de la evaluación de los archivos 4 a 6

```
Top coincidencias para el archivo sospechoso 'FID-07.txt':  
- org-041.txt con una similitud del 80.59%  
  
Top coincidencias para el archivo sospechoso 'FID-08.txt':  
- org-079.txt con una similitud del 94.30%  
  
Top coincidencias para el archivo sospechoso 'FID-09.txt':  
- org-109.txt con una similitud del 92.98%  
- org-082.txt con una similitud del 20.95%  
- org-103.txt con una similitud del 20.14%  
  
Top coincidencias para el archivo sospechoso 'FID-10.txt':  
- org-007.txt con una similitud del 76.05%  
- org-001.txt con una similitud del 34.27%  
- org-002.txt con una similitud del 30.62%  
- org-010.txt con una similitud del 27.61%  
- org-006.txt con una similitud del 23.54%  
  
Top coincidencias para el archivo sospechoso 'FID-11.txt':  
  
El tiempo de ejecución fue de : 1.2876999378204346
```

Imagen . Resultados de la evaluación de los archivos 7 a 11 y tiempo de ejecución

Es importante señalar que el archivo “FID-11.txt” es un archivo extra creado con la finalidad de mostrar al código archivos que no contienen plagio alguno y así ver el comportamiento del modelo hacia archivos con plagio y archivos si plagio

```
True Positives: 10  
False Positives: 0  
True Negatives: 1  
False Negatives: 0  
Precision: 1.00  
Recall: 1.00  
Accuracy: 1.00  
F1 Score: 1.00  
AUC (ROC): 1.00
```

Imagen . Resultados de las métricas establecidas primeros 10 archivos

```
True Positives: 22  
False Positives: 1  
True Negatives: 7  
False Negatives: 1  
Precision: 0.96  
Recall: 0.96  
Accuracy: 0.94  
F1 Score: 0.96  
AUC (ROC): 0.81
```

Imagen . Resultados de las métricas establecidas con 30 archivos

Conclusiones

Se ha logrado implementar exitosamente un programa que calcula la similitud entre un documento y una serie de documentos considerados originales. Este programa demuestra una eficacia notable al identificar plagios que emplean técnicas básicas; sin embargo, enfrenta desafíos al detectar parafraseos y otras técnicas más sofisticadas de alteración textual, lo que resulta en una reducción de precisión en la detección de plagio.

Para superar estas limitaciones, se recomienda explorar el uso de modelos de procesamiento de lenguaje natural (NLP) pre entrenados, como BERT o ROBERTA, que han mostrado capacidad para identificar variaciones complejas en los textos, incluyendo parafraseos. Adicionalmente, la implementación de modelos avanzados de inteligencia artificial, como las Máquinas de Vectores de Soporte (SVC), junto con la generación de embeddings de los documentos, podría mejorar significativamente la detección al revelar patrones de plagio más sutiles y complejos. Estas mejoras no solo aumentan la precisión del programa sino que también ampliarían su aplicabilidad en escenarios de detección de plagio más desafiantes.

Anexos

Repositorio de GitHub:

https://github.com/A01745336/Evidencia1_Detecci-nDePlagio