9. **Detector de plagio básico:** Compara dos archivos de texto para encontrar similitudes.

¡Claro! Puedo ayudarte con una aplicación básica en Python para detectar plagio comparando dos archivos de texto.
Este script utilizará una técnica sencilla: **comparar la proporción de palabras que son comunes** entre los dos documentos, ignorando la diferencia en la longitud total de los textos. Es un enfoque muy básico, pero efectivo para encontrar similitudes obvias.

? Detector de Plagio Básico en Python

Aquí tienes el código de Python. Lo he dividido en una función para leer archivos y otra para calcular la similitud.

Python

```
import string

def leer_y_normalizar_archivo(ruta_archivo):
    """
    Lee un archivo, convierte el texto a minúsculas y elimina la puntuación.
    Retorna un conjunto de palabras únicas para facilitar la comparación.
    """
    try:
        with open(ruta_archivo, 'r', encoding='utf-8') as f:
            texto = f.read()
    except FileNotFoundError:
        print(f"Error: No se encontró el archivo en la ruta '{ruta_archivo}'.")
        return None

    # 1. Convertir a minúsculas
    texto = texto.lower()

    # 2. Eliminar puntuación
    texto = texto.translate(str.maketrans('', '', string.punctuation))

    # 3. Dividir en palabras y crear un conjunto (para palabras únicas)
    palabras = texto.split()

    return set(palabras)

def calcular_similitud(palabras1, palabras2):
    """
    Calcula el porcentaje de palabras comunes entre dos conjuntos.
    Usa la fórmula: (Palabras Comunes / Palabras Totales Únicas) * 100
    """
    # Si alguno de los archivos no pudo leerse, retornamos 0
    if palabras1 is None or palabras2 is None:
        return 0.0

    # 1. Encontrar las palabras comunes (intersección de conjuntos)
    palabras_comunes = palabras1.intersection(palabras2)

    # 2. Encontrar todas las palabras únicas (unión de conjuntos)
```

```
    palabras_totales_unicas = palabras1.union(palabras2)

    # Manejar el caso de archivos vacíos para evitar división por cero
    if not palabras_totales_unicas:
        return 0.0

    # 3. Calcular el índice de similitud (Jaccard Index)
    similitud = (len(palabras_comunes) / len(palabras_totales_unicas)) * 100

    return similitud

# --- Función Principal ---
def detector_plagio_basico(ruta1, ruta2):
    """
    Ejecuta el proceso de detección de plagio para dos archivos.
    """
    print(f"? Analizando similitudes entre '{ruta1}' y '{ruta2}'...")

    palabras_doc1 = leer_y_normalizar_archivo(ruta1)
    palabras_doc2 = leer_y_normalizar_archivo(ruta2)

    if palabras_doc1 is None or palabras_doc2 is None:
        return

    similitud_porcentaje = calcular_similitud(palabras_doc1, palabras_doc2)

    print("\n--- Resultados del Análisis ---")
    print(f"Palabras únicas en el Documento 1: **{len(palabras_doc1)}**")
    print(f"Palabras únicas en el Documento 2: **{len(palabras_doc2)}**")
    print("-" * 30)
    print(f"⭐ Índice de Similitud (Básico): **{similitud_porcentaje:.2f}%**")

    if similitud_porcentaje >= 50:
        print("? ALERTA: La alta similitud sugiere un posible plagio o un origen común.")
    elif similitud_porcentaje >= 20:
        print(" Advertencia: Hay una similitud notable. Se recomienda una revisión más profunda.")
    else:
        print("✅ Similitud baja. Los documentos parecen ser distintos.")


# --- Ejemplo de Uso ---
#  IMPORTANTE: Asegúrate de que estos archivos existan en la misma carpeta
# que tu script de Python, o cambia las rutas por las correctas.

# Crea dos archivos de prueba (ejemplo: 'original.txt' y 'copia.txt') para probar:
# Documento 1: El sol brilla. El cielo es azul. La hierba es verde.
# Documento 2: El sol brilla. El cielo es muy azul. La hierba es verde, muy verde.

if __name__ == "__main__":
    ruta_archivo_original = 'documento_original.txt'
    ruta_archivo_sospechoso = 'documento_sospechoso.txt'

    # Puedes cambiar estas rutas para usar tus propios archivos
    detector_plagio_basico(ruta_archivo_original, ruta_archivo_sospechoso)
```

? Cómo Usarlo

1. **Guarda el Código:** Guarda el código anterior en un archivo llamado, por ejemplo, detector.py.
2. **Crea Archivos de Prueba:** En la *misma carpeta* que detector.py, crea dos archivos de texto:
- documento_original.txt
- documento_sospechoso.txt
3. **Ejecuta el Script:** Abre tu terminal o línea de comandos, navega hasta la carpeta y ejecuta:
Bash
python detector.py

? Limitaciones y Mejoras

El método que utiliza este script se llama **Índice de Jaccard** y es muy simple.

Limitación	¿Qué Ignora?
Orden de Palabras	Una frase como "El perro muerde al hombre" y "El hombre muerde al perro" tendrían una similitud del **100%** de palabras únicas, aunque el significado es totalmente diferente.
Stop Words	No elimina palabras comunes (artículos, preposiciones: *el, la, un, y, de, a*), lo que artificialmente aumenta la similitud en textos largos.
Estructura	No compara frases completas o párrafos.

Para una detección de plagio más robusta, se requerirían técnicas más avanzadas, como:

- **N-gramas:** Comparar secuencias de N palabras (*"el perro", "perro muerde", "muerde al"*). Esto detecta plagio incluso si se cambia ligeramente el orden.
- **Vectorización (TF-IDF):** Convertir el texto en vectores numéricos y usar algoritmos de distancia (como la **similitud coseno**) para una comparación semántica más profunda.
¿Te gustaría que implemente una versión más avanzada usando N-gramas o una función para descargar y analizar documentos PDF?

¡Me encantaría ayudarte a implementarlo!
Para una detección de plagio más efectiva, el método de **N-gramas** es el siguiente paso lógico. Los N-gramas nos permiten comparar no solo palabras sueltas, sino **secuencias de palabras**, lo que es mucho mejor para capturar la estructura de las frases plagiadas.

? Detector de Plagio con N-gramas

Este script mejorado:

1. **Normaliza el texto** (minúsculas, sin puntuación).
2. **Genera secuencias de palabras** (N-gramas). Usaremos un N de 3, lo que significa que compararemos todas las secuencias de tres palabras seguidas.
3. **Calcula la similitud Jaccard** sobre estos N-gramas.
Python

```
import string

def leer_y_normalizar_archivo(ruta_archivo):
    """
    Lee un archivo, convierte el texto a minúsculas y elimina la puntuación.
    Retorna una lista de palabras.
    """
    try:
        with open(ruta_archivo, 'r', encoding='utf-8') as f:
            texto = f.read()
    except FileNotFoundError:
```

```
        print(f"❌ Error: No se encontró el archivo en la ruta '{ruta_archivo}'.")
        return None

    # 1. Convertir a minúsculas
    texto = texto.lower()

    # 2. Eliminar puntuación
    texto = texto.translate(str.maketrans('', '', string.punctuation))

    # 3. Dividir en palabras (lista)
    palabras = texto.split()

    return palabras

def generar_n_gramas(palabras, n=3):
    """
    Convierte una lista de palabras en un conjunto de N-gramas (secuencias de N palabras).

    Ejemplo: si n=3 y la lista es ['el', 'sol', 'brilla', 'fuerte']
    Los 3-gramas serán: {'el sol brilla', 'sol brilla fuerte'}
    """
    if len(palabras) < n:
        return set() # El texto es muy corto para generar N-gramas

    n_gramas = []
    # Itera desde la primera palabra hasta la N-ésima palabra antes del final
    for i in range(len(palabras) - n + 1):
        # Toma el slice de 'n' palabras, las une con un espacio
        grama = ' '.join(palabras[i:i + n])
        n_gramas.append(grama)

    # Retorna un conjunto de N-gramas únicos para la comparación
    return set(n_gramas)

def calcular_similitud_n_gramas(n_gramas1, n_gramas2):
    """
    Calcula el porcentaje de N-gramas comunes entre dos conjuntos (Índice de Jaccard).
    """
    if n_gramas1 is None or n_gramas2 is None:
        return 0.0

    # 1. Encontrar los N-gramas comunes (intersección)
    comunes = n_gramas1.intersection(n_gramas2)

    # 2. Encontrar todos los N-gramas únicos (unión)
    totales_unicos = n_gramas1.union(n_gramas2)

    if not totales_unicos:
        return 0.0

    # 3. Calcular la similitud
    similitud = (len(comunes) / len(totales_unicos)) * 100

    return similitud
```

```
# --- Función Principal ---
def detector_plagio_n_gramas(ruta1, ruta2, n=3):
    """
    Ejecuta el proceso de detección de plagio usando N-gramas.
    """
    print(f"? Analizando similitudes (con {n}-gramas) entre '{ruta1}' y '{ruta2}'...")

    # 1. Leer y normalizar
    palabras_doc1 = leer_y_normalizar_archivo(ruta1)
    palabras_doc2 = leer_y_normalizar_archivo(ruta2)

    if palabras_doc1 is None or palabras_doc2 is None:
        return

    # 2. Generar N-gramas
    n_gramas_doc1 = generar_n_gramas(palabras_doc1, n)
    n_gramas_doc2 = generar_n_gramas(palabras_doc2, n)

    # 3. Calcular similitud
    similitud_porcentaje = calcular_similitud_n_gramas(n_gramas_doc1, n_gramas_doc2)

    print("\n--- Resultados del Análisis (N-gramas) ---")
    print(f"Número de {n}-gramas en Documento 1: **{len(n_gramas_doc1)}**")
    print(f"Número de {n}-gramas en Documento 2: **{len(n_gramas_doc2)}**")
    print("-" * 40)
    print(f"⭐ Índice de Similitud de {n}-gramas: **{similitud_porcentaje:.2f}%**")

    if similitud_porcentaje >= 70:
        print("? ALERTA: La similitud de frases es extremadamente alta, indicando plagio directo.")
    elif similitud_porcentaje >= 30:
        print(" Advertencia: Hay una similitud notable en la estructura de las frases. Se recomienda revisión.")
    else:
        print("✅ Similitud estructural baja. Los documentos parecen ser independientes.")


# --- Ejemplo de Uso ---
#  IMPORTANTE: Asegúrate de que los archivos existan en la misma carpeta.

if __name__ == "__main__":
    # N=3 (3-gramas) es un buen valor inicial, compara secuencias de 3 palabras.
    N_VALUE = 3

    # Puedes usar los mismos nombres de archivo que en el ejemplo anterior
    ruta_archivo_original = 'documento_original.txt'
    ruta_archivo_sospechoso = 'documento_sospechoso.txt'

    # Prueba la nueva función mejorada
    detector_plagio_n_gramas(ruta_archivo_original, ruta_archivo_sospechoso, n=N_VALUE)
```

Ejemplo Práctico de 3-gramas

Si los documentos dicen:

Documento 1 **El perro muerde al hombre.**

Documento 2

El **perro muerde al** gato.

3-gramas D1	**3-gramas D2**
el perro muerde	el perro muerde
perro muerde al	perro muerde al
muerde al hombre	muerde al gato

- **Comunes:** 2
- **Totales Únicos:** 4
- **Similitud:** $(2/4) \times 100 = 50\%$

Este método es **más sensible** a los cambios en el texto que la simple comparación de palabras sueltas.