El diseño de un sistema para analizar, clasificar y calificar documentos requiere una combinación de técnicas de \*\*procesamiento de lenguaje natural (NLP)\*\*, \*\*aprendizaje automático (Machine Learning)\*\* y un enfoque estructurado para evaluar el contenido. A continuación, detallo cómo se puede implementar:

---

### \*\*1. Arquitectura del Sistema\*\*

#### \*\*A. Captura de Documentos\*\*

- \*\*Entrada:\*\* Archivos en formato PDF, Word, Excel.

- \*\*Extracción de Contenido:\*\*

- PDF: Librerías como `PyPDF2`, `pdfminer` o `PyMuPDF`.

- Word: Librerías como `python-docx`.

- Excel: Librerías como `openpyxl` o `pandas`.

#### \*\*B. Análisis de Contenido\*\*

1. \*\*Procesamiento de Texto:\*\*

- Limpieza y tokenización del texto.

- Eliminación de palabras irrelevantes (stop words).

- Identificación de palabras clave o temas mediante técnicas como TF-IDF o embeddings de texto (por ejemplo, con modelos preentrenados como BERT).

2. \*\*Clasificación Inicial:\*\*

- Uso de modelos supervisados para clasificar los documentos en:

- \*\*Público\*\*

- \*\*Privado\*\*

- \*\*Secreto\*\*

- Algoritmos sugeridos:

- SVM

- Random Forest

- Redes neuronales como Transformer (BERT).

3. \*\*Score o Puntaje:\*\*

- Calcular un score basado en:

- Sensibilidad del lenguaje.

- Presencia de palabras clave sensibles.

- Naturaleza del contenido (datos personales, contratos, información financiera, etc.).

---

### \*\*2. Clasificación Automática Mejorada por IA\*\*

1. \*\*Entrenamiento Inicial:\*\*

- Recopilar y etiquetar manualmente un conjunto de documentos para entrenar el modelo de clasificación.

- Utilizar embeddings preentrenados para contexto lingüístico (ej. `spaCy`, `transformers` de Hugging Face).

2. \*\*Aprendizaje Activo:\*\*

- El sistema solicita al usuario confirmar o corregir clasificaciones ambiguas.

- Se agregan los documentos etiquetados al conjunto de entrenamiento, mejorando el modelo.

3. \*\*Modelo de Reentrenamiento:\*\*

- Configurar un pipeline automático para reentrenar periódicamente el modelo con nuevos datos.

---

### \*\*3. Infraestructura\*\*

- \*\*Procesamiento y Almacenamiento:\*\*

- Utilizar servicios como AWS S3 o Azure Blob Storage para almacenar los documentos.

- Bases de datos para almacenar metadatos y resultados de clasificación (MongoDB o PostgreSQL).

- \*\*Pipeline de Datos:\*\*

- Uso de herramientas como Apache Kafka o Airflow para manejar el flujo de datos.

- \*\*Interfaz:\*\*

- Crear una interfaz web (usando frameworks como Flask o FastAPI) para cargar documentos, visualizar resultados y gestionar la clasificación.

---

### \*\*4. Ejemplo de Flujo Técnico\*\*

#### \*\*A. Cargar y Analizar Documento\*\*

```python

from PyPDF2 import PdfReader

from transformers import pipeline

# Cargar y extraer texto de un PDF

reader = PdfReader("documento.pdf")

texto = " ".join([page.extract\_text() for page in reader.pages])

# Modelo de clasificación de texto

clasificador = pipeline("text-classification", model="bert-base-uncased")

resultado = clasificador(texto)

print(resultado)

```

#### \*\*B. Clasificación por Nivel de Sensibilidad\*\*

```python

# Definir etiquetas

etiquetas = {

"público": 0.1,

"privado": 0.5,

"secreto": 0.9

}

# Calcular puntaje

puntaje = sum(etiquetas.get(token, 0) for token in texto.split())

nivel = "secreto" if puntaje > 0.7 else "privado" if puntaje > 0.3 else "público"

print(f"Clasificación: {nivel}, Puntaje: {puntaje}")

```

---

### \*\*5. Mejora Continua\*\*

- \*\*Feedback del Usuario:\*\*

- Una interfaz para que los usuarios corrijan clasificaciones erróneas.

- \*\*Exploración de Modelos Avanzados:\*\*

- Incorporar modelos como `OpenAI GPT` para un análisis más contextual.

- \*\*Registro y Auditoría:\*\*

- Registrar decisiones del sistema para evaluar su precisión con el tiempo.

---

### \*\*6. Consideraciones de Seguridad\*\*

- \*\*Cifrado:\*\*

- Cifrar documentos almacenados y en tránsito.

- \*\*Gestión de Accesos:\*\*

- Control de roles para usuarios que gestionan los documentos.

- \*\*Privacidad:\*\*

- Asegurar que el sistema cumpla con regulaciones como GDPR o CCPA.

---

Este diseño modular permite un sistema robusto que puede escalar y mejorar con el tiempo gracias a técnicas de aprendizaje continuo. ¿Deseas profundizar en algún aspecto en particular o implementar un prototipo?

Crear un sistema que dado un punto de inicio pueda analizar, separar, clasificar y colocar un scoore de los documentos pdf, word, excel, los mismo que serán clasificados en público, privado y secreto, también que pueda ir aprendiendo con IA, para mejorar su clasificación

**ChatGPT dijo:**

ChatGPT

El diseño de un sistema para analizar, clasificar y calificar documentos requiere una combinación de técnicas de **procesamiento de lenguaje natural (NLP)**, **aprendizaje automático (Machine Learning)** y un enfoque estructurado para evaluar el contenido. A continuación, detallo cómo se puede implementar:

**1. Arquitectura del Sistema**

**A. Captura de Documentos**

* **Entrada:** Archivos en formato PDF, Word, Excel.
* **Extracción de Contenido:**
  + PDF: Librerías como PyPDF2, pdfminer o PyMuPDF.
  + Word: Librerías como python-docx.
  + Excel: Librerías como openpyxl o pandas.

**B. Análisis de Contenido**

1. **Procesamiento de Texto:**
   * Limpieza y tokenización del texto.
   * Eliminación de palabras irrelevantes (stop words).
   * Identificación de palabras clave o temas mediante técnicas como TF-IDF o embeddings de texto (por ejemplo, con modelos preentrenados como BERT).
2. **Clasificación Inicial:**
   * Uso de modelos supervisados para clasificar los documentos en:
     + **Público**
     + **Privado**
     + **Secreto**
   * Algoritmos sugeridos:
     + SVM
     + Random Forest
     + Redes neuronales como Transformer (BERT).
3. **Score o Puntaje:**
   * Calcular un score basado en:
     + Sensibilidad del lenguaje.
     + Presencia de palabras clave sensibles.
     + Naturaleza del contenido (datos personales, contratos, información financiera, etc.).

**2. Clasificación Automática Mejorada por IA**

1. **Entrenamiento Inicial:**
   * Recopilar y etiquetar manualmente un conjunto de documentos para entrenar el modelo de clasificación.
   * Utilizar embeddings preentrenados para contexto lingüístico (ej. spaCy, transformers de Hugging Face).
2. **Aprendizaje Activo:**
   * El sistema solicita al usuario confirmar o corregir clasificaciones ambiguas.
   * Se agregan los documentos etiquetados al conjunto de entrenamiento, mejorando el modelo.
3. **Modelo de Reentrenamiento:**
   * Configurar un pipeline automático para reentrenar periódicamente el modelo con nuevos datos.

**3. Infraestructura**

* **Procesamiento y Almacenamiento:**
  + Utilizar servicios como AWS S3 o Azure Blob Storage para almacenar los documentos.
  + Bases de datos para almacenar metadatos y resultados de clasificación (MongoDB o PostgreSQL).
* **Pipeline de Datos:**
  + Uso de herramientas como Apache Kafka o Airflow para manejar el flujo de datos.
* **Interfaz:**
  + Crear una interfaz web (usando frameworks como Flask o FastAPI) para cargar documentos, visualizar resultados y gestionar la clasificación.

**4. Ejemplo de Flujo Técnico**

**A. Cargar y Analizar Documento**

python

Copiar código

from PyPDF2 import PdfReader

from transformers import pipeline

# Cargar y extraer texto de un PDF

reader = PdfReader("documento.pdf")

texto = " ".join([page.extract\_text() for page in reader.pages])

# Modelo de clasificación de texto

clasificador = pipeline("text-classification", model="bert-base-uncased")

resultado = clasificador(texto)

print(resultado)

**B. Clasificación por Nivel de Sensibilidad**

python

Copiar código

# Definir etiquetas

etiquetas = {

"público": 0.1,

"privado": 0.5,

"secreto": 0.9

}

# Calcular puntaje

puntaje = sum(etiquetas.get(token, 0) for token in texto.split())

nivel = "secreto" if puntaje > 0.7 else "privado" if puntaje > 0.3 else "público"

print(f"Clasificación: {nivel}, Puntaje: {puntaje}")

**5. Mejora Continua**

* **Feedback del Usuario:**
  + Una interfaz para que los usuarios corrijan clasificaciones erróneas.
* **Exploración de Modelos Avanzados:**
  + Incorporar modelos como OpenAI GPT para un análisis más contextual.
* **Registro y Auditoría:**
  + Registrar decisiones del sistema para evaluar su precisión con el tiempo.

**6. Consideraciones de Seguridad**

* **Cifrado:**
  + Cifrar documentos almacenados y en tránsito.
* **Gestión de Accesos:**
  + Control de roles para usuarios que gestionan los documentos.
* **Privacidad:**
  + Asegurar que el sistema cumpla con regulaciones como GDPR o CCPA.

Este diseño modular permite un sistema robusto que puede escalar y mejorar con el tiempo gracias a técnicas de aprendizaje continuo. ¿Deseas profundizar en algún aspecto en particular o implementar un prototipo?

4o

**Dijiste:**

ayúdame a corregir el error del punto B File ... line 9, in <module> puntaje = sum(etiquetas.get(token, 0) for token in texto.split()) ^^^^^ NameError: name 'texto' is not defined. Did you mean: 'next'?

**ChatGPT dijo:**

ChatGPT

El error indica que la variable texto no está definida en el momento en que se ejecuta la línea de código donde se calcula puntaje. Esto puede deberse a que el bloque que define texto no se ejecutó correctamente o no fue incluido en el código.

A continuación, presento una versión corregida y funcional del código:

**Código Corregido**

python

Copiar código

from PyPDF2 import PdfReader

# Extraer texto de un archivo PDF

try:

# Ruta del archivo PDF

ruta\_pdf = "documento.pdf"

reader = PdfReader(ruta\_pdf)

# Extraer texto de todas las páginas del PDF

texto = " ".join([page.extract\_text() for page in reader.pages])

print("Texto extraído correctamente.")

except FileNotFoundError:

print(f"Error: El archivo '{ruta\_pdf}' no existe.")

texto = "" # Si falla, texto se inicializa vacío

# Definir etiquetas y sus pesos

etiquetas = {

"público": 0.1,

"privado": 0.5,

"secreto": 0.9

}

# Calcular puntaje basado en el contenido del texto

if texto: # Asegurarse de que texto no esté vacío

puntaje = sum(etiquetas.get(token.lower(), 0) for token in texto.split())

nivel = (

"secreto" if puntaje > 0.7

else "privado" if puntaje > 0.3

else "público"

)

print(f"Clasificación: {nivel}, Puntaje: {puntaje}")

else:

print("No se pudo clasificar el documento debido a un error en la extracción de texto.")

**Explicación de la Corrección**

1. **Asegurar la Definición de texto:**
   * Se verifica que el texto sea extraído correctamente del archivo PDF. Si el archivo no existe o hay un error en la lectura, texto se inicializa como una cadena vacía ("").
2. **Control de Excepciones:**
   * Se agrega un manejo de excepciones para casos en los que el archivo PDF no se encuentre (FileNotFoundError).
3. **Validación Antes del Análisis:**
   * Antes de calcular el puntaje, el programa comprueba si texto no está vacío para evitar errores adicionales.
4. **Compatibilidad con Mayúsculas/Minúsculas:**
   * Se usa token.lower() para asegurar que los tokens coincidan con las claves de etiquetas, independientemente de las mayúsculas.

**Resultado Esperado**

Si el archivo documento.pdf está presente y contiene texto, el programa debería:

1. Extraer el texto.
2. Calcular el puntaje basado en las etiquetas definidas.
3. Imprimir la clasificación y el puntaje.

Si el archivo no está presente o no contiene texto, el programa informará del error sin fallar.

4o

**Dijiste:**

Adicionar en el código que pueda leer todo un directorio con archivos .pdf, word, excel y texto

**ChatGPT dijo:**

ChatGPT

Aquí tienes una versión del código que puede procesar todos los archivos en un directorio, incluyendo **PDFs**, **documentos de Word**, **Excel** y **archivos de texto**. El programa extraerá el texto de cada archivo, calculará el puntaje y clasificará cada documento.

**Código Mejorado**

python

Copiar código

import os

from PyPDF2 import PdfReader

from docx import Document

import pandas as pd

# Función para leer texto de archivos PDF

def leer\_pdf(ruta):

try:

reader = PdfReader(ruta)

texto = " ".join([page.extract\_text() for page in reader.pages])

return texto

except Exception as e:

print(f"Error leyendo PDF '{ruta}': {e}")

return ""

# Función para leer texto de archivos Word

def leer\_word(ruta):

try:

doc = Document(ruta)

texto = " ".join([p.text for p in doc.paragraphs])

return texto

except Exception as e:

print(f"Error leyendo Word '{ruta}': {e}")

return ""

# Función para leer texto de archivos Excel

def leer\_excel(ruta):

try:

# Leer todas las hojas y concatenar contenido

df = pd.concat(pd.read\_excel(ruta, sheet\_name=None), ignore\_index=True)

texto = " ".join(df.astype(str).stack()) # Convertir todo a texto y concatenar

return texto

except Exception as e:

print(f"Error leyendo Excel '{ruta}': {e}")

return ""

# Función para leer texto de archivos de texto (.txt)

def leer\_txt(ruta):

try:

with open(ruta, "r", encoding="utf-8") as file:

return file.read()

except Exception as e:

print(f"Error leyendo TXT '{ruta}': {e}")

return ""

# Función para procesar un archivo y clasificarlo

def procesar\_archivo(ruta, etiquetas):

extension = os.path.splitext(ruta)[1].lower()

if extension == ".pdf":

texto = leer\_pdf(ruta)

elif extension in [".docx", ".doc"]:

texto = leer\_word(ruta)

elif extension in [".xlsx", ".xls"]:

texto = leer\_excel(ruta)

elif extension == ".txt":

texto = leer\_txt(ruta)

else:

print(f"Tipo de archivo no soportado: {ruta}")

return

# Calcular puntaje y clasificación

if texto:

puntaje = sum(etiquetas.get(token.lower(), 0) for token in texto.split())

nivel = (

"secreto" if puntaje > 0.7

else "privado" if puntaje > 0.3

else "público"

)

print(f"Archivo: {ruta}\nClasificación: {nivel}, Puntaje: {puntaje}\n")

else:

print(f"No se pudo extraer texto del archivo: {ruta}")

# Función para procesar todos los archivos en un directorio

def procesar\_directorio(directorio, etiquetas):

for raiz, \_, archivos in os.walk(directorio):

for archivo in archivos:

ruta\_completa = os.path.join(raiz, archivo)

procesar\_archivo(ruta\_completa, etiquetas)

# Etiquetas y pesos para la clasificación

etiquetas = {

"público": 0.1,

"privado": 0.5,

"secreto": 0.9

}

# Ruta del directorio a procesar

directorio = "ruta\_a\_tu\_directorio"

# Procesar todos los archivos en el directorio

procesar\_directorio(directorio, etiquetas)

**Explicación del Código**

1. **Soporte para Múltiples Tipos de Archivos:**
   * **PDF:** Usa PyPDF2.
   * **Word:** Usa python-docx.
   * **Excel:** Usa pandas para leer todas las hojas y concatenar el contenido.
   * **TXT:** Usa lectura directa con open.
2. **Procesamiento de Archivos en un Directorio:**
   * Usa os.walk para recorrer todos los archivos en el directorio y sus subdirectorios.
   * Identifica el tipo de archivo por su extensión y llama a la función correspondiente para extraer texto.
3. **Clasificación:**
   * Después de extraer el texto, se calcula un puntaje basado en las etiquetas definidas.
   * Se asigna una categoría (público, privado, secreto) en función del puntaje.
4. **Manejo de Errores:**
   * Control de excepciones para evitar que errores en un archivo detengan el procesamiento completo.

**Ejemplo de Ejecución**

* Si tienes un directorio con archivos como:
  + documento1.pdf
  + archivo2.docx
  + datos3.xlsx
  + nota4.txt

El programa leerá cada archivo, extraerá su texto, calculará el puntaje y mostrará la clasificación correspondiente.

**Salida Esperada**

plaintext

Copiar código

Archivo: /ruta/documento1.pdf

Clasificación: privado, Puntaje: 0.45

Archivo: /ruta/archivo2.docx

Clasificación: secreto, Puntaje: 0.85

Archivo: /ruta/datos3.xlsx

Clasificación: público, Puntaje: 0.10

Archivo: /ruta/nota4.txt

Clasificación: privado, Puntaje: 0.50

**Personalización**

Puedes ajustar las etiquetas y los pesos en el diccionario etiquetas para adaptarlo a tus necesidades específicas. ¿Te gustaría agregar alguna funcionalidad adicional?

4o

**Dijiste:**

arreglar el error Traceback (most recent call last): File "D:\CLASIFICADOR\clasificaNivel.py", line 3, in <module> from docx import Document File "C:\Users\Oscar\PycharmProjects\pythonProject\.venv\Lib\site-packages\docx.py", line 30, in <module> from exceptions import PendingDeprecationWarning ModuleNotFoundError: No module named 'exceptions'

**ChatGPT dijo:**

ChatGPT

El error se produce porque hay un conflicto con un archivo llamado docx.py en el entorno de tu proyecto que está sobrescribiendo el módulo oficial python-docx. Este conflicto puede surgir si tienes un archivo con el mismo nombre (docx.py) en tu proyecto o en una ubicación en el **PYTHONPATH**.

**Soluciones:**

**1. Verifica el Archivo Conflictivo**

1. Busca si existe un archivo llamado docx.py en tu proyecto o directorio de trabajo actual.
   * Por ejemplo, en D:\CLASIFICADOR o en cualquier carpeta asociada.
2. Si lo encuentras, renómbralo (por ejemplo, mi\_docx.py) o elimínalo si no lo necesitas.

**2. Reinstala el Módulo Correcto**

Es posible que el módulo python-docx no esté instalado correctamente o haya sido sobrescrito. Puedes reinstalarlo con el siguiente comando:

bash

Copiar código

pip uninstall docx python-docx -y

pip install python-docx

* **Nota:** Usa python-docx, ya que el paquete docx es una versión obsoleta y no oficial.

**3. Verifica el Entorno Virtual**

Si estás utilizando un entorno virtual, asegúrate de que esté activado antes de instalar python-docx:

bash

Copiar código

# Activar entorno virtual (en Windows)

venv\Scripts\activate

# Instalar python-docx

pip install python-docx

**4. Verifica Importaciones en tu Código**

Cambia la línea de importación en tu código a:

python

Copiar código

from docx import Document

Si este error persiste después de los pasos anteriores, verifica las versiones de los módulos instalados con:

bash

Copiar código

pip list

Busca python-docx en la lista y asegúrate de que esté instalado.

**Verifica si el Problema Persiste**

Después de realizar las correcciones, intenta ejecutar el script nuevamente. Si tienes más problemas, indícalos para que pueda ayudarte a solucionarlos. 😊

4o

**Dijiste:**

Corregir el error File "D:\CLASIFICADOR\clasificaNivel.py", line 88 directorio = "C:\Users\Oscar\Downloads" ^^^^^^^^^^^^^^^^^^^^^^^^^^ SyntaxError: (unicode error) 'unicodeescape' codec can't decode bytes in position 2-3: truncated \UXXXXXXXX escape

**ChatGPT dijo:**

ChatGPT

El error ocurre porque las rutas en Python que contienen el carácter **\** son interpretadas como secuencias de escape (por ejemplo, \U se considera el inicio de un "Unicode escape"). Para solucionarlo, puedes usar uno de los siguientes métodos:

**Soluciones**

**1. Usar Literales Raw (r"")**

Coloca un prefijo r antes de la cadena para indicarle a Python que interprete la cadena como "cruda" (raw string), donde los caracteres de escape no tienen un significado especial:

python

Copiar código

directorio = r"C:\Users\Oscar\Downloads"

**2. Usar Doble Barra Invertida**

Escapa cada barra invertida (\) convirtiéndola en una doble barra invertida (\\). Esto evita que Python las interprete como secuencias de escape:

python

Copiar código

directorio = "C:\\Users\\Oscar\\Downloads"

**3. Usar Barras Normales /**

En Python, las barras normales / funcionan correctamente en las rutas de Windows. Esto es compatible y evita problemas de escape:

python

Copiar código

directorio = "C:/Users/Oscar/Downloads"

**Ejemplo Completo Corregido**

Si estás usando el código anterior:

python

Copiar código

# Ruta del directorio a procesar

directorio = r"C:\Users\Oscar\Downloads"

# Procesar todos los archivos en el directorio

procesar\_directorio(directorio, etiquetas)

Cualquiera de las tres soluciones funcionará sin errores. Si encuentras algún problema adicional, avísame. 😊

4o

**Dijiste:**

Ahora ponemos en una interface gráfica con un panel con django

2/2

**ChatGPT dijo:**

ChatGPT

Agregar una interfaz gráfica para este clasificador usando **Django** implica crear un proyecto web donde puedas cargar archivos, procesarlos y mostrar sus clasificaciones. A continuación, te muestro los pasos para implementar esta funcionalidad:

**1. Crear el Proyecto Django**

1. **Instalar Django:**

bash

Copiar código

pip install django

1. **Crear un Proyecto Django:**

bash

Copiar código

django-admin startproject clasificador

cd clasificador

1. **Crear una Aplicación Django:**

bash

Copiar código

python manage.py startapp documentos

1. **Registrar la Aplicación en settings.py:** En el archivo clasificador/settings.py, agrega 'documentos' a la lista INSTALLED\_APPS:

python

Copiar código

INSTALLED\_APPS = [

...

'documentos',

]

**2. Configurar el Modelo y Formulario**

En documentos/models.py, define un modelo para manejar los archivos cargados:

python

Copiar código

from django.db import models

class Documento(models.Model):

archivo = models.FileField(upload\_to="archivos/")

clasificacion = models.CharField(max\_length=50, blank=True)

puntaje = models.FloatField(null=True, blank=True)

cargado\_en = models.DateTimeField(auto\_now\_add=True)

def \_\_str\_\_(self):

return self.archivo.name

**Crear un Formulario para Subir Archivos**

En documentos/forms.py:

python

Copiar código

from django import forms

from .models import Documento

class DocumentoForm(forms.ModelForm):

class Meta:

model = Documento

fields = ['archivo']

**3. Crear la Lógica de Clasificación**

En documentos/utils.py, implementa las funciones para procesar y clasificar archivos:

python

Copiar código

import os

from PyPDF2 import PdfReader

from docx import Document

import pandas as pd

# Funciones de lectura según el tipo de archivo

def leer\_pdf(ruta):

try:

reader = PdfReader(ruta)

return " ".join([page.extract\_text() for page in reader.pages])

except Exception as e:

return ""

def leer\_word(ruta):

try:

doc = Document(ruta)

return " ".join([p.text for p in doc.paragraphs])

except Exception as e:

return ""

def leer\_excel(ruta):

try:

df = pd.concat(pd.read\_excel(ruta, sheet\_name=None), ignore\_index=True)

return " ".join(df.astype(str).stack())

except Exception as e:

return ""

def leer\_txt(ruta):

try:

with open(ruta, "r", encoding="utf-8") as file:

return file.read()

except Exception as e:

return ""

# Clasificación del archivo

def clasificar\_archivo(archivo):

etiquetas = {"público": 0.1, "privado": 0.5, "secreto": 0.9}

extension = os.path.splitext(archivo)[1].lower()

if extension == ".pdf":

texto = leer\_pdf(archivo)

elif extension in [".docx", ".doc"]:

texto = leer\_word(archivo)

elif extension in [".xlsx", ".xls"]:

texto = leer\_excel(archivo)

elif extension == ".txt":

texto = leer\_txt(archivo)

else:

return "no soportado", 0

puntaje = sum(etiquetas.get(token.lower(), 0) for token in texto.split())

nivel = "secreto" if puntaje > 0.7 else "privado" if puntaje > 0.3 else "público"

return nivel, puntaje

**4. Crear las Vistas**

En documentos/views.py, define las vistas para manejar el formulario y procesar archivos:

python

Copiar código

from django.shortcuts import render, redirect

from .forms import DocumentoForm

from .models import Documento

from .utils import clasificar\_archivo

def cargar\_archivo(request):

if request.method == "POST":

form = DocumentoForm(request.POST, request.FILES)

if form.is\_valid():

documento = form.save()

archivo\_path = documento.archivo.path

clasificacion, puntaje = clasificar\_archivo(archivo\_path)

documento.clasificacion = clasificacion

documento.puntaje = puntaje

documento.save()

return redirect('listar\_documentos')

else:

form = DocumentoForm()

return render(request, 'cargar\_archivo.html', {'form': form})

def listar\_documentos(request):

documentos = Documento.objects.all()

return render(request, 'listar\_documentos.html', {'documentos': documentos})

**5. Crear las Plantillas**

1. **Plantilla para Cargar Archivos (templates/cargar\_archivo.html):**

html

Copiar código

<!DOCTYPE html>

<html>

<head>

<title>Subir Archivo</title>

</head>

<body>

<h1>Subir Archivo</h1>

<form method="post" enctype="multipart/form-data">

{% csrf\_token %}

{{ form.as\_p }}

<button type="submit">Subir</button>

</form>

<a href="{% url 'listar\_documentos' %}">Ver Documentos</a>

</body>

</html>

1. **Plantilla para Listar Documentos (templates/listar\_documentos.html):**

html

Copiar código

<!DOCTYPE html>

<html>

<head>

<title>Documentos Clasificados</title>

</head>

<body>

<h1>Documentos Clasificados</h1>

<table border="1">

<tr>

<th>Archivo</th>

<th>Clasificación</th>

<th>Puntaje</th>

<th>Cargado En</th>

</tr>

{% for documento in documentos %}

<tr>

<td>{{ documento.archivo.name }}</td>

<td>{{ documento.clasificacion }}</td>

<td>{{ documento.puntaje }}</td>

<td>{{ documento.cargado\_en }}</td>

</tr>

{% endfor %}

</table>

<a href="{% url 'cargar\_archivo' %}">Subir Nuevo Archivo</a>

</body>

</html>

**6. Configurar URLs**

En documentos/urls.py:

python

Copiar código

from django.urls import path

from .views import cargar\_archivo, listar\_documentos

urlpatterns = [

path('cargar/', cargar\_archivo, name='cargar\_archivo'),

path('listar/', listar\_documentos, name='listar\_documentos'),

]

En clasificador/urls.py, incluye las rutas de la aplicación:

python

Copiar código

from django.contrib import admin

from django.urls import path, include

urlpatterns = [

path('admin/', admin.site.urls),

path('', include('documentos.urls')),

]

**7. Migrar y Ejecutar el Proyecto**

1. Aplica migraciones para el modelo:

bash

Copiar código

python manage.py makemigrations

python manage.py migrate

1. Ejecuta el servidor:

bash

Copiar código

python manage.py runserver

**Prueba**

* Accede a http://127.0.0.1:8000/cargar/ para subir un archivo.
* Visita http://127.0.0.1:8000/listar/ para ver la clasificación de los archivos cargados.

¿Te gustaría agregar algo más, como filtros o exportación de resultados? 😊

4o

**Dijiste:**

Me ayudas a solucionar el error

**ChatGPT dijo:**

ChatGPT

¡Por supuesto! Por favor, indícame el error exacto que estás enfrentando al ejecutar el proyecto Django para poder ayudarte de forma específica. Si tienes un **traceback** o mensaje de error, compártelo conmigo para identificar el problema y solucionarlo rápidamente. 😊

4o

**Dijiste:**

Internal Server Error: /cargar/ Traceback (most recent call last): File "C:\Users\Oscar\PycharmProjects\pythonProject\.venv\Lib\site-packages\django\core\handlers\exception.py", line 55, in inner response = get\_response(request) ^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Oscar\PycharmProjects\pythonProject\.venv\Lib\site-packages\django\core\handlers\base.py", line 197, in \_get\_response response = wrapped\_callback(request, \*callback\_args, \*\*callback\_kwargs) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "D:\CLASIFICADOR\clasificador\documentos\views.py", line 19, in cargar\_archivo return render(request, 'cargar\_archivo.html', {'form': form}) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Oscar\PycharmProjects\pythonProject\.venv\Lib\site-packages\django\shortcuts.py", line 25, in render content = loader.render\_to\_string(template\_name, context, request, using=using) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Oscar\PycharmProjects\pythonProject\.venv\Lib\site-packages\django\template\loader.py", line 61, in render\_to\_string template = get\_template(template\_name, using=using) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Oscar\PycharmProjects\pythonProject\.venv\Lib\site-packages\django\template\loader.py", line 19, in get\_template raise TemplateDoesNotExist(template\_name, chain=chain) django.template.exceptions.TemplateDoesNotExist: cargar\_archivo.html

**ChatGPT dijo:**

ChatGPT

El error **TemplateDoesNotExist** indica que Django no puede encontrar el archivo de plantilla cargar\_archivo.html. Esto suele suceder por problemas de ubicación o configuración. Vamos a solucionarlo.

**1. Verifica la Ubicación del Archivo de Plantilla**

Asegúrate de que el archivo cargar\_archivo.html esté ubicado en la carpeta de plantillas de tu aplicación documentos. Sigue estos pasos:

1. Crea un directorio llamado templates dentro de tu aplicación documentos:

markdown

Copiar código

documentos/

templates/

cargar\_archivo.html

1. Si tienes varios archivos de plantillas, organiza las plantillas dentro de una subcarpeta con el nombre de tu aplicación:

markdown

Copiar código

documentos/

templates/

documentos/

cargar\_archivo.html

Esto es una práctica recomendada para evitar conflictos de nombres en proyectos grandes.

**2. Asegúrate de que Django Reconozca las Plantillas**

En el archivo settings.py, verifica que la configuración TEMPLATES tenga configurada la opción APP\_DIRS como True:

python

Copiar código

TEMPLATES = [

{

'BACKEND': 'django.template.backends.django.DjangoTemplates',

'DIRS': [], # Puedes especificar rutas adicionales aquí si tienes plantillas personalizadas.

'APP\_DIRS': True, # Esto debe estar en True.

'OPTIONS': {

'context\_processors': [

'django.template.context\_processors.debug',

'django.template.context\_processors.request',

'django.contrib.auth.context\_processors.auth',

'django.contrib.messages.context\_processors.messages',

],

},

},

]

**3. Verifica el Nombre del Archivo**

* Asegúrate de que el archivo se llama exactamente cargar\_archivo.html (incluyendo las minúsculas y la extensión .html).
* Evita extensiones dobles o caracteres extraños como cargar\_archivo.html.html.

**4. Reinicia el Servidor**

Después de corregir la ubicación y el nombre del archivo, reinicia el servidor Django:

bash

Copiar código

python manage.py runserver

**5. Prueba Nuevamente**

Accede a la URL http://127.0.0.1:8000/cargar/ y verifica si la página se carga correctamente.

**Si el Problema Persiste**

Si después de estos pasos el problema continúa, revisemos juntos la estructura del proyecto y la configuración. 😊

4o

**Dijiste:**

Quisiera que se pueda subir todo un directorio o desde un punto de inicio

**ChatGPT dijo:**

ChatGPT

Subir todo un directorio o procesar archivos desde un punto de inicio implica extender la lógica para aceptar múltiples archivos. Esto puede lograrse usando un enfoque basado en subir un archivo comprimido (por ejemplo, un ZIP) o especificar un directorio en el servidor para que Django lo procese. Aquí te presento cómo implementar ambas opciones:

**Opción 1: Subir un Archivo ZIP con Todo el Directorio**

En esta opción, el usuario sube un archivo ZIP que contiene múltiples archivos, y Django lo descomprime y procesa.

**Cambios en el Código**

1. **Actualizar el Modelo (models.py)** No es necesario cambiar el modelo, ya que se procesará el contenido del ZIP directamente.
2. **Actualizar el Formulario (forms.py)** Permitir la subida de un archivo ZIP:

python

Copiar código

from django import forms

class ArchivoZipForm(forms.Form):

archivo\_zip = forms.FileField(label="Subir un archivo ZIP")

1. **Actualizar la Vista para Procesar el ZIP (views.py)** Descomprimir y procesar los archivos dentro del ZIP:

python

Copiar código

import os

import zipfile

from django.shortcuts import render, redirect

from .forms import ArchivoZipForm

from .utils import clasificar\_archivo # Función existente para clasificar archivos

from .models import Documento

def cargar\_directorio(request):

if request.method == "POST":

form = ArchivoZipForm(request.POST, request.FILES)

if form.is\_valid():

archivo\_zip = request.FILES['archivo\_zip']

ruta\_temporal = "temp/" # Carpeta temporal para almacenar archivos descomprimidos

os.makedirs(ruta\_temporal, exist\_ok=True)

# Guardar el ZIP temporalmente

zip\_path = os.path.join(ruta\_temporal, archivo\_zip.name)

with open(zip\_path, 'wb') as f:

for chunk in archivo\_zip.chunks():

f.write(chunk)

# Descomprimir el ZIP

with zipfile.ZipFile(zip\_path, 'r') as zip\_ref:

zip\_ref.extractall(ruta\_temporal)

# Procesar archivos extraídos

for root, dirs, files in os.walk(ruta\_temporal):

for file in files:

archivo\_path = os.path.join(root, file)

clasificacion, puntaje = clasificar\_archivo(archivo\_path)

Documento.objects.create(

archivo=file,

clasificacion=clasificacion,

puntaje=puntaje

)

# Limpiar archivos temporales

for root, dirs, files in os.walk(ruta\_temporal, topdown=False):

for file in files:

os.remove(os.path.join(root, file))

for dir in dirs:

os.rmdir(os.path.join(root, dir))

os.rmdir(ruta\_temporal)

return redirect('listar\_documentos')

else:

form = ArchivoZipForm()

return render(request, 'cargar\_directorio.html', {'form': form})

1. **Crear la Plantilla para Subir un ZIP** Archivo templates/cargar\_directorio.html:

html

Copiar código

<!DOCTYPE html>

<html>

<head>

<title>Subir Directorio</title>

</head>

<body>

<h1>Subir Directorio (ZIP)</h1>

<form method="post" enctype="multipart/form-data">

{% csrf\_token %}

{{ form.as\_p }}

<button type="submit">Subir</button>

</form>

<a href="{% url 'listar\_documentos' %}">Ver Documentos Clasificados</a>

</body>

</html>

1. **Agregar la URL** En documentos/urls.py:

python

Copiar código

from .views import cargar\_directorio

urlpatterns += [

path('cargar\_directorio/', cargar\_directorio, name='cargar\_directorio'),

]

**Opción 2: Procesar Archivos Desde un Directorio en el Servidor**

En este caso, el usuario especifica una ruta base en el servidor donde Django procesa los archivos automáticamente.

**Cambios en el Código**

1. **Crear una Vista para Procesar un Directorio** Agrega esta función en views.py:

python

Copiar código

import os

from django.shortcuts import render

from .utils import clasificar\_archivo

from .models import Documento

def procesar\_directorio(request):

if request.method == "POST":

ruta\_inicial = request.POST.get('ruta\_base')

if ruta\_inicial and os.path.exists(ruta\_inicial):

for root, dirs, files in os.walk(ruta\_inicial):

for file in files:

archivo\_path = os.path.join(root, file)

clasificacion, puntaje = clasificar\_archivo(archivo\_path)

Documento.objects.create(

archivo=file,

clasificacion=clasificacion,

puntaje=puntaje

)

return redirect('listar\_documentos')

return render(request, 'procesar\_directorio.html')

1. **Crear la Plantilla para Ingresar la Ruta** Archivo templates/procesar\_directorio.html:

html

Copiar código

<!DOCTYPE html>

<html>

<head>

<title>Procesar Directorio</title>

</head>

<body>

<h1>Procesar Archivos desde un Directorio</h1>

<form method="post">

{% csrf\_token %}

<label for="ruta\_base">Ruta del Directorio:</label>

<input type="text" id="ruta\_base" name="ruta\_base" required>

<button type="submit">Procesar</button>

</form>

<a href="{% url 'listar\_documentos' %}">Ver Documentos Clasificados</a>

</body>

</html>

1. **Agregar la URL** En documentos/urls.py:

python

Copiar código

from .views import procesar\_directorio

urlpatterns += [

path('procesar\_directorio/', procesar\_directorio, name='procesar\_directorio'),

]

**Conclusión**

Con estas opciones:

* **Opción 1 (ZIP):** Permite subir un archivo comprimido con todo un directorio.
* **Opción 2 (Ruta Base):** Procesa todos los archivos de un directorio ya existente en el servidor.

¿Te gustaría ayuda para implementar alguna opción en particular o ambas? 😊

4o

**Dijiste:**

Qué tal si adicionamos una interface más bonita en los formularios y tablas con bootstrap

**ChatGPT dijo:**

ChatGPT

¡Agregar Bootstrap es una excelente idea para mejorar la apariencia de los formularios y tablas en tu proyecto Django! Aquí tienes los pasos para integrar Bootstrap y estilizar tus vistas.

**1. Instalar Bootstrap en tu Proyecto**

1. **Incluir Bootstrap en las Plantillas** Usa la CDN de Bootstrap. Modifica tus plantillas base para incluir los enlaces a los estilos de Bootstrap.

Crea un archivo base.html en la carpeta templates:

html

Copiar código

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>{% block title %}Clasificador{% endblock %}</title>

<!-- Bootstrap CSS -->

<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha3/dist/css/bootstrap.min.css" rel="stylesheet">

</head>

<body class="bg-light">

<nav class="navbar navbar-expand-lg navbar-dark bg-dark">

<div class="container">

<a class="navbar-brand" href="#">Clasificador</a>

<div class="collapse navbar-collapse" id="navbarNav">

<ul class="navbar-nav">

<li class="nav-item"><a class="nav-link" href="{% url 'cargar\_archivo' %}">Subir Archivo</a></li>

<li class="nav-item"><a class="nav-link" href="{% url 'listar\_documentos' %}">Ver Documentos</a></li>

<li class="nav-item"><a class="nav-link" href="{% url 'cargar\_directorio' %}">Subir Directorio</a></li>

</ul>

</div>

</div>

</nav>

<div class="container mt-4">

{% block content %}{% endblock %}

</div>

<!-- Bootstrap JS -->

<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha3/dist/js/bootstrap.bundle.min.js"></script>

</body>

</html>

1. Extiende base.html en tus plantillas para reutilizar el diseño. Por ejemplo, en cargar\_archivo.html:

html

Copiar código

{% extends "base.html" %}

{% block title %}Subir Archivo{% endblock %}

{% block content %}

<div class="card">

<div class="card-body">

<h2 class="card-title">Subir Archivo</h2>

<form method="post" enctype="multipart/form-data" class="mt-3">

{% csrf\_token %}

{{ form.as\_p }}

<button type="submit" class="btn btn-primary">Subir</button>

</form>

<a href="{% url 'listar\_documentos' %}" class="btn btn-link mt-2">Ver Documentos Clasificados</a>

</div>

</div>

{% endblock %}

**2. Mejorar las Tablas**

Estiliza la tabla en listar\_documentos.html con clases de Bootstrap:

html

Copiar código

{% extends "base.html" %}

{% block title %}Documentos Clasificados{% endblock %}

{% block content %}

<h2>Documentos Clasificados</h2>

<table class="table table-striped table-bordered mt-3">

<thead class="table-dark">

<tr>

<th>Archivo</th>

<th>Clasificación</th>

<th>Puntaje</th>

<th>Cargado En</th>

</tr>

</thead>

<tbody>

{% for documento in documentos %}

<tr>

<td>{{ documento.archivo.name }}</td>

<td>{{ documento.clasificacion }}</td>

<td>{{ documento.puntaje }}</td>

<td>{{ documento.cargado\_en }}</td>

</tr>

{% endfor %}

</tbody>

</table>

<a href="{% url 'cargar\_archivo' %}" class="btn btn-primary mt-3">Subir Nuevo Archivo</a>

{% endblock %}

**3. Estilizar Otros Formularios**

Para otros formularios como subir un directorio, aplica las mismas clases de Bootstrap en sus plantillas:

html

Copiar código

{% extends "base.html" %}

{% block title %}Subir Directorio{% endblock %}

{% block content %}

<div class="card">

<div class="card-body">

<h2 class="card-title">Subir Directorio</h2>

<form method="post" enctype="multipart/form-data" class="mt-3">

{% csrf\_token %}

<div class="mb-3">

<label for="archivo\_zip" class="form-label">Seleccionar Archivo ZIP:</label>

{{ form.archivo\_zip }}

</div>

<button type="submit" class="btn btn-primary">Procesar</button>

</form>

<a href="{% url 'listar\_documentos' %}" class="btn btn-link mt-2">Ver Documentos Clasificados</a>

</div>

</div>

{% endblock %}

**4. Opcional: Añadir Alertas para Feedback**

Puedes mostrar alertas para acciones exitosas o errores. En tu vista, usa mensajes de Django:

En views.py:

python

Copiar código

from django.contrib import messages

def cargar\_archivo(request):

if request.method == "POST":

form = DocumentoForm(request.POST, request.FILES)

if form.is\_valid():

documento = form.save()

archivo\_path = documento.archivo.path

clasificacion, puntaje = clasificar\_archivo(archivo\_path)

documento.clasificacion = clasificacion

documento.puntaje = puntaje

documento.save()

messages.success(request, "Archivo subido y clasificado con éxito.")

return redirect('listar\_documentos')

else:

messages.error(request, "Error al procesar el archivo. Por favor, inténtelo de nuevo.")

else:

form = DocumentoForm()

return render(request, 'cargar\_archivo.html', {'form': form})

En base.html, muestra las alertas:

html

Copiar código

{% if messages %}

<div class="container mt-3">

{% for message in messages %}

<div class="alert alert-{{ message.tags }}">

{{ message }}

</div>

{% endfor %}

</div>

{% endif %}

**Resultado**

Con Bootstrap, tu interfaz será más atractiva y funcional:

* **Formularios:** estilizados con campos alineados y botones destacados.
* **Tablas:** mejor presentación, colores alternados y bordes visibles.
* **Navegación:** una barra superior para acceder rápidamente a las opciones.

¿Te gustaría agregar algo más, como iconos o gráficos? 😊