

Tecnológico Nacional de México / Instituto Tecnológico de Culiacán.



**TECNOLÓGICO
NACIONAL DE MÉXICO**



Ingeniería en Sistemas Computacionales.

Inteligencia Artificial

Zuriel Dathan Mora Felix.

Modulo 2

Detección de Spam

Ontiveros Sánchez Jesús Daniel (22170750).

Ramírez Ruiz Santiago (22170783).

Horario de 9:00 a 10:00 AM.

Culiacán Sinaloa a 12 de octubre del 2025

Documentación de Código de Detección de Spam

Archivo DeteccionDeSpam.py

Es el archivo donde se realizó la parte del backend para hacer los cálculos las funciones de limpieza, calcular probabilidades, etc.

Librerías utilizadas

```
import os, re, unicodedata          # Importa módulos estándar: manejo
de sistema, regex y normalización Unicode
from pathlib import Path             # Permite manejar rutas de archivos
de forma multiplataforma
import numpy as np, pandas as pd, nltk # Importa librerías: NumPy
(matemática), pandas (dataframes), NLTK (texto)
from sklearn.feature_extraction.text import TfidfVectorizer # Convierte
texto en vectores TF-IDF
```

Función de limpieza del texto

```
# ----- FUNCIONES DE LIMPIEZA -----

def quitar_acentos(t: str) -> str:      # Quita acentos de un texto dado
    t = unicodedata.normalize('NFKD', t) # Normaliza el texto separando
letras y acentos
    return t.encode('ascii', 'ignore').decode('utf-8') # Convierte a ASCII
eliminando acentos

def limpiar_texto(t: str) -> str:      # Limpia el texto para procesamiento
    t = (t or "").lower().strip()      # Convierte a minúsculas y elimina
espacios extremos
    t = quitar_acentos(t)              # Llama a quitar_acentos()
    t = re.sub(r"^[a-z0-9\s]", " ", t) # Sustituye todo lo que no sea letra
o número por espacio
    return re.sub(r"\s+", " ", t).strip() # Reemplaza múltiples espacios por
uno solo
```

Estas funciones preparan el texto para el análisis: eliminan acentos, mayúsculas, caracteres raros y espacios innecesarios.

Esto garantiza que el modelo procese solo palabras limpias y uniformes.

Funciones de Extracción

```
# ----- FUNCIONES DE EXTRACCIÓN -----

def dominio_de_email(remitente: str) -> str:      # Extrae el dominio del
correo (después del @)
    r = (remitente or "").strip().lower()         # Limpia y pone en
minúsculas el remitente
    if "@" in r:                                  # Verifica que contenga
"@"
        return r.split("@", 1)[-1]              # Retorna el dominio
(partido después del @)
    return ""                                     # Si no, retorna cadena
vacía

def tokens_dominio(dom: str) -> list[str]:        # Convierte dominio en
tokens útiles
    if not dom:                                   # Si está vacío, devuelve
lista vacía
        return []
    dom = dom.replace("-", " ")                  # Reemplaza guiones por
espacios
    partes = dom.split(".")                      # Separa dominio por
puntos
    toks = []                                    # Inicializa lista de
tokens
    if len(partes) >= 1:                         # Si hay al menos un
segmento
        toks.append(f"from_dom_{limpiar_texto(partes[-2] if len(partes)>=2
else partes[0])}") # Token dominio
    if len(partes) >= 2:                         # Si hay TLD
        toks.append(f"from_tld_{limpiar_texto(partes[-1])}") # Token del
TLD (.com, .org)
    if len(partes) >= 3:                         # Si hay subdominio
        toks.append(f"from_sub_{limpiar_texto(partes[0])}") # Token
subdominio
    return toks                                  # Retorna la lista de
tokens

def extraer_enlaces(texto: str) -> list[str]:    # Extrae URLs del texto
    return URL_RE.findall(texto or "")           # Aplica expresión
regular definida en Config_regex

def tokens_enlace(url: str) -> list[str]:        # Convierte una URL en
tokens
```

```

    toks = ["has_url"] # Token que indica
presencia de URL
    try: # Control de errores
        m = re.search(r"https?:\/\/([^\s:]+)", url, re.I) # Busca el dominio
dentro de la URL
        if not m: # Si no encuentra,
retorna token base
            return toks
        host = m.group(1).lower() # Obtiene dominio
principal
        host = host.replace("www.", "") # Elimina "www."
        partes = host.split(".") # Separa por puntos
        if len(partes) >= 1: # Token de dominio
            toks.append(f"url_dom_{limpiar_texto(partes[-2] if
len(partes)>=2 else partes[0])}")
        if len(partes) >= 2: # Token de TLD
            toks.append(f"url_tld_{limpiar_texto(partes[-1])}")
        if len(partes) >= 3: # Token de subdominio
            toks.append(f"url_sub_{limpiar_texto(partes[0])}")
    except Exception: # Si ocurre error, ignora
        pass
    return toks # Devuelve lista de
tokens

def extraer_adjuntos(texto: str) -> list[tuple[str,str]]: # Extrae nombres
y extensiones de adjuntos
    adjuntos = [] # Lista vacía
    for m in ADJUNTO_RE.finditer(texto or ""): # Busca coincidencias
usando regex
        nombre = (m.group("name") or "").strip() # Obtiene nombre del
archivo
        ext = (m.group("ext") or "").lower().strip() # Obtiene extensión
        if nombre and ext: # Si ambos existen
            adjuntos.append((nombre, ext)) # Añade como tupla
(nombre, extensión)
    return adjuntos # Devuelve lista de
adjuntos

def tokens_adjuntos(adjuntos: list[tuple[str,str]]) -> list[str]: #
Convierte adjuntos a tokens
    toks = [] # Lista vacía
    if adjuntos: # Si hay adjuntos
        toks.append("has_attachment") # Marca que hay adjuntos
    for nombre, ext in adjuntos: # Recorre cada adjunto
        toks.append(f"att_ext_{ext}") # Token con la extensión

```

```

        if ext in EXT_PELIGROSAS:                # Si la extensión es
del adjunto peligrosa
            toks.append("att_ext_dangerous")      # Añade token de peligro
            base = limpiar_texto(re.sub(r"\.[a-z0-9]{1,6}$", "", nombre,
flags=re.I)) # Limpia nombre base
            if base:                             # Si hay texto
                for w in base.split():            # Divide en palabras
                    if len(w) >= 3:              # Evita palabras cortas
                        toks.append(f"att_name_{w}") # Añade token con nombre
            return toks                          # Retorna lista final

```

Estas funciones extraen partes importantes del correo:

- Dominio del remitente (por ejemplo, “gmail.com”).
- Enlaces en el cuerpo del mensaje.
- Archivos adjuntos y sus extensiones.

Convierten esos datos en tokens (palabras clave) que el modelo usa para aprender patrones típicos del spam, como dominios falsos o extensiones peligrosas (.bat, .cmd, .exe).

Clase EmailSpamClassifier

```

# ----- CLASE PRINCIPAL -----

class EmailSpamClassifier:                        # Define clase principal
del clasificador
    def __init__(self, csv_path=None):           # Constructor con ruta
opcional al dataset
        try:
            nltk.data.find("corpora/stopwords") # Verifica corpus de
stopwords
        except LookupError:
            nltk.download("stopwords")          # Descarga si no está
        try:
            nltk.data.find("tokenizers/punkt")  # Verifica tokenizer
        except LookupError:
            nltk.download("punkt")              # Descarga si falta

        self.precision = 0.0                   # Inicializa precisión
        self.recall_spam = 0.0                 # Inicializa recall para
spam

```

```

        if csv_path is None:                                # Si no se pasa CSV
            try:
                base = Path(__file__).resolve().parent      # Usa ruta del
archivo actual
            except NameError:
                base = Path(os.getcwd())                    # O directorio actual
            csv_path = base / "datasets" / "spam_ham_dataset2.csv" # Ruta
por defecto

        self.csv_path = Path(csv_path)                      # Guarda ruta del CSV

        if self.csv_path.exists():                          # Si el archivo existe
            df = pd.read_csv(self.csv_path)                 # Carga dataset
        else:                                                # Si no existe, usa
dataset de ejemplo
            df = pd.DataFrame({
                "remitente": [
                    "promos@casino-oro.fun",
                    "maria.garcia@empresa.com",
                    "soporte@seguridadbancaria-alerta.com",
                    "contabilidad@empresa.com",
                ],
                "asunto": [
                    "100 tiradas gratis sin depósito",
                    "Reunión semanal del equipo",
                    "Verifica tu cuenta urgente",
                    "Reporte mensual",
                ],
                "mensaje": [
                    "Regístrate y gana premios al instante. premios.bat", #
archivo .bat peligroso
                    "Nos vemos mañana 10:00 sala de juntas. reporte.pdf",
                    "Detectamos actividad inusual. verificar-cuenta.cmd", #
archivo .cmd peligroso
                    "Envío el reporte mensual consolidado. balance.xlsx",
                ],
                "enlaces": [
                    "https://casino-oro.fun/oferta",
                    "",
                    "https://seguridadbancaria-alerta.com/verificar-cuenta",
                    "",
                ],
                "etiqueta": ["spam", "ham", "spam", "ham"], # Etiquetas
reales
            })

```

```

        if "etiqueta" not in df.columns:  # Verifica existencia de
columna etiqueta
            raise ValueError("El CSV debe contener la columna 'etiqueta' con
valores 'spam' o 'ham'.")

        df["remitente"] = df.get("remitente", "").astype(str)  # Convierte
remitente a texto
        df["asunto"] = df.get("asunto", "").astype(str)  # Convierte
asunto a texto
        df["mensaje"] = df.get("mensaje", "").astype(str)  # Convierte
mensaje a texto
        df["enlaces"] = df.get("enlaces", "").astype(str)  # Convierte
enlaces a texto
        df["etiqueta"] =
df["etiqueta"].astype(str).str.lower().str.strip()  # Limpia etiquetas

        rows = []  # Lista de textos
enriquecidos
        for _, r in df.iterrows():  # Itera sobre filas del
DataFrame
            remitente = r["remitente"]  # Obtiene remitente
            asunto = r["asunto"]  # Obtiene asunto
            mensaje = r["mensaje"]  # Obtiene mensaje
            enlaces_col = r["enlaces"]  # Obtiene enlaces

            enlaces_list = []  # Inicializa lista de
enlaces

            if isinstance(enlaces_col, str) and enlaces_col.strip():  # Si
hay texto en columna enlaces
                enlaces_list = [u for u in enlaces_col.split() if
u.strip()]  # Separa por espacios
                enlaces_list = list(set(enlaces_list +
extraer_enlaces(mensaje)))  # Añade enlaces del mensaje

            adjuntos = extraer_adjuntos(mensaje)  # Extrae adjuntos del
mensaje

            enriched = self._make_feature_text(remitente, asunto, mensaje,
enlaces_list, adjuntos)  # Genera texto
            rows.append(enriched)  # Añade texto enriquecido

        df["mensaje_limpio"] = rows  # Añade columna procesada
        self.df = df  # Guarda DataFrame

```

```

        self.vectorizer = TfidfVectorizer(ngram_range=(1, 2), min_df=1) #
Crea vectorizador TF-IDF
        X = self.vectorizer.fit_transform(self.df["mensaje_limpio"]) #
Ajusta y transforma corpus
        self.palabras = self.vectorizer.get_feature_names_out() #
Guarda vocabulario

        spam = self.df[self.df["etiqueta"] == "spam"] # Filtra spam
        ham = self.df[self.df["etiqueta"] == "ham"] # Filtra ham
        n = len(self.df) or 1 # Número total de
muestras
        self.P_spam = len(spam) / n # Probabilidad a
priori de spam
        self.P_no_spam = len(ham) / n # Probabilidad a
priori de ham

        X_spam = self.vectorizer.transform(spam["mensaje_limpio"]) if
len(spam) else X[:0] # Vectoriza spam
        X_ham = self.vectorizer.transform(ham["mensaje_limpio"]) if
len(ham) else X[:0] # Vectoriza ham

        alpha = 1.0 # Suavizado de Laplace
        s_spam = np.sum(X_spam.toarray(), axis=0) if X_spam.shape[0] else
np.zeros(len(self.palabras)) # Suma frecuencias spam
        s_ham = np.sum(X_ham.toarray(), axis=0) if X_ham.shape[0] else
np.zeros(len(self.palabras)) # Suma frecuencias ham

        denom_spam = (np.sum(s_spam) + alpha * len(self.palabras)) or 1.0 #
Denominador spam
        denom_ham = (np.sum(s_ham) + alpha * len(self.palabras)) or 1.0 #
Denominador ham
        self.P_feat_spam = (s_spam + alpha) / denom_spam #
Probabilidades condicionales spam
        self.P_feat_ham = (s_ham + alpha) / denom_ham #
Probabilidades condicionales ham

        try: # Calcula precisión y
recall
            self.df["prediccion"] =
self.df["mensaje_limpio"].apply(self.clasificar_texto)
            self.precision = float(np.mean(self.df["prediccion"] ==
self.df["etiqueta"]))
            denom = self.df["etiqueta"].value_counts().get("spam", 1)
            self.recall_spam = float(

```



```

        np.sum((self.df["prediccion"] == "spam") &
(self.df["etiqueta"] == "spam")) / denom
    )
    except Exception:
        pass # Ignora errores

```

Esta clase contiene todo el funcionamiento principal del detector de spam.

Cuando se crea una instancia, el constructor:

- Carga el dataset de correos (real o de ejemplo).
- Limpia los textos con las funciones anteriores.
- Aplica TF-IDF para transformar las palabras en números.
- Calcula probabilidades de aparición de cada palabra en correos SPAM y HAM.
- Entrena un modelo basado en la Teoría de Bayes.

Funciones de Teto Enriquecido

```

# ===== FEATURE TEXT =====

# ===== TEXTO ENRIQUECIDO =====
def _make_feature_text(self, remitente: str, asunto: str, contenido:
str,
                        enlaces: list[str] | None, adjuntos:
list[tuple[str,str]] | None) -> str:
    parts: list[str] = [] # Lista donde se irán
    acumulando las partes del texto enriquecido
    is_dangerous = False # Bandera opcional (no
    usada directamente aquí)

    # Texto base (limpio)
    parts.append(limpiar_texto(asunto)) # Limpia y añade el
    asunto
    parts.append(limpiar_texto(contenido)) # Limpia y añade el
    contenido del mensaje

    # Dominio remitente -> tokens
    dom = dominio_de_email(remitente) # Obtiene el dominio
    del remitente
    parts += tokens_dominio(dom) # Genera tokens del
    dominio y los agrega a la lista

```

```

        # Enlaces → tokens
        if enlaces:
            # Si se proporcionan
            enlaces explícitamente
            for u in enlaces:
                # Recorre cada enlace
                parts += tokens_enlace(u)
            # Extrae tokens del
            enlace
        else:
            # Si no se pasaron
            enlaces, los busca dentro del contenido
            for u in extraer_enlaces(contenido):
                # Extrae enlaces
                encontrados en el cuerpo del mensaje
                parts += tokens_enlace(u)
            # Los convierte a
            tokens

        # Adjuntos → tokens
        if adjuntos is None:
            # Si no se pasaron
            adjuntos explícitamente
            adjuntos = extraer_adjuntos(contenido)
            # Los extrae del
            contenido
            adj_tokens = tokens_adjuntos(adjuntos)
            # Convierte los
            adjuntos a tokens
            parts += adj_tokens
            # Agrega los tokens de
            adjuntos a la lista

        # Refuerzo para tokens peligrosos: Aumentar el peso (importancia)
        if "att_ext_dangerous" in adj_tokens:
            # Si hay un adjunto con
            extensión peligrosa
            parts += ["att_ext_dangerous"] * 5
            # Repite el token
            varias veces para aumentar su peso TF-IDF

        # Unir todo
        return " ".join([p for p in parts if p])
        # Une todos los tokens
        en una sola cadena separada por espacios

```

Aquí se reúne toda la información del correo (asunto, cuerpo, enlaces, adjuntos, dominio, etc.) en un solo texto.

Si detecta un archivo peligroso, repite el token varias veces para aumentar su peso en el modelo TF-IDF.

Este texto final representa las “características” que usará el modelo para clasificar.

Funciones para calcular mediante la Teoría de Bayes

```
# ===== NÚCLEO BAYES =====

def _log_post(self, txt_clean: str):          # Calcula las
probabilidades logarítmicas de SPAM y HAM
    v = self.vectorizer.transform([txt_clean]).toarray()[0] # Convierte
el texto en vector TF-IDF
    eps = 1e-12                                # Valor pequeño para
evitar log(0)
    ls = np.log(self.P_spam + eps) + np.sum(np.log(self.P_feat_spam +
eps) * v) # Log probabilidad de SPAM
    lh = np.log(self.P_no_spam + eps) + np.sum(np.log(self.P_feat_ham +
eps) * v) # Log probabilidad de HAM
    return ls, lh                                # Devuelve ambas
probabilidades logarítmicas

def clasificar_texto(self, txt: str) -> str:  # Clasifica un texto
cualquiera (sin formato de correo)
    t = limpiar_texto(txt)                    # Limpia el texto
    ls, lh = self._log_post(t)                # Calcula las
probabilidades logarítmicas
    return "spam" if ls > lh else "ham"        # Retorna "spam" si su
log-probabilidad es mayor

def prob_spam_texto(self, txt: str) -> float: # Devuelve probabilidad
numérica de ser SPAM
    t = limpiar_texto(txt)                    # Limpia el texto
    ls, lh = self._log_post(t)                # Obtiene log-
probabilidades
    d = lh - ls                                # Diferencia entre
log(HAM) y log(SPAM)
    return float(1.0 / (1.0 + np.exp(d)))      # Aplica sigmoide ->
valor entre 0 y 1 (probabilidad)
```

Calcula las probabilidades logarítmicas de que un mensaje pertenezca a SPAM o HAM según los tokens detectados.

El modelo compara ambos valores:

- Si la probabilidad de SPAM es mayor → clasifica como SPAM.
- Si no → es HAM (correo legítimo).

La función clasificar texto incluye una regla de decisión la que calcula cual compara las dos probabilidades (SPAM vs HAM) y devuelve la etiqueta más probable según el modelo.

Funciones para correo

```
# ===== FUNCIONES PARA CORREO =====

    def clasificar_correo(self, remitente: str, asunto: str, contenido: str)
-> str:
    """
        Compatibilidad retro: extrae enlaces/adjuntos automáticamente del
        contenido.
    """
    enriched = self._make_feature_text(remitente, asunto, contenido,
None, None) # Genera texto enriquecido
    ls, lh = self._log_post(enriched) # Calcula
    probabilidades logarítmicas
    return "spam" if ls > lh else "ham" # Devuelve etiqueta
    según probabilidad mayor

    def prob_spam_correo(self, remitente: str, asunto: str, contenido: str)
-> float:
    enriched = self._make_feature_text(remitente, asunto, contenido,
None, None) # Texto enriquecido
    ls, lh = self._log_post(enriched) # Probabilidades
    logarítmicas
    d = lh - ls # Diferencia
    logarítmica
    return float(1.0 / (1.0 + np.exp(d))) # Devuelve
    probabilidad de ser SPAM

    # Versión extendida (puedes pasar enlaces y adjuntos ya extraídos desde
    la UI)
    def clasificar_correo_ext(self, remitente: str, asunto: str, contenido:
    str,
                                enlaces: list[str] | None = None,
                                adjuntos: list[tuple[str,str]] | None = None)
-> str:
    enriched = self._make_feature_text(remitente, asunto, contenido,
    enlaces, adjuntos) # Texto enriquecido con todo
    ls, lh = self._log_post(enriched) # Calcula
    probabilidades logarítmicas
    return "spam" if ls > lh else "ham" # Clasifica como
    SPAM o HAM

    def prob_spam_correo_ext(self, remitente: str, asunto: str, contenido:
    str,
```

```

                                enlaces: list[str] | None = None,
                                adjuntos: list[tuple[str,str]] | None = None) -
> float:
    enriched = self._make_feature_text(remitente, asunto, contenido,
enlaces, adjuntos) # Texto enriquecido completo
    ls, lh = self._log_post(enriched)                # Calcula
probabilidades
    d = lh - ls                                     # Diferencia entre
HAM y SPAM
    return float(1.0 / (1.0 + np.exp(d)))           # Retorna
probabilidad entre 0 y 1 de ser SPAM

```

Esta sección del programa contiene las funciones encargadas de analizar y clasificar los correos electrónicos ingresados por el usuario o por una interfaz gráfica.

En otras palabras, actúa como una interfaz directa entre el modelo de detección de spam y el usuario final, permitiendo determinar si un correo es legítimo (HAM) o no deseado (SPAM) a partir de los datos proporcionados.

El método principal, `clasificar_correo()`, recibe como parámetros el remitente, el asunto y el contenido del correo.

Internamente, esta función extrae automáticamente los enlaces y archivos adjuntos del texto y genera un texto enriquecido mediante la función `_make_feature_text()`, que combina todos los elementos relevantes del mensaje (dominio, enlaces, palabras clave, extensiones de archivos, etc.).

Luego, con ayuda del método `_log_post()`, el sistema calcula las probabilidades logarítmicas de que el mensaje pertenezca a la clase SPAM o HAM.

Finalmente, compara ambos valores y devuelve la etiqueta final (“spam” o “ham”) según la probabilidad más alta.

De forma complementaria, la función `prob_spam_correo()` realiza el mismo proceso, pero devuelve un valor numérico entre 0 y 1, que representa la probabilidad de que el mensaje sea spam.

Esto se logra aplicando una función matemática sigmoide que transforma las probabilidades en un número interpretativo.

Por ejemplo, un valor cercano a 1 indica alta probabilidad de ser SPAM, mientras que uno próximo a 0 sugiere que el correo es legítimo.

El programa también incluye versiones extendidas de ambas funciones:

`clasificar_correo_ext()` y `prob_spam_correo_ext()`.

Estas permiten pasar manualmente los enlaces y adjuntos ya detectados desde una interfaz externa, evitando que el programa los busque nuevamente.

Archivo de Config_regex.py

```
# Configuraciones de expresiones regulares

import re # Importa el módulo estándar para trabajar con expresiones
regulares en Python

# --- Detecta URLs (enlaces) ---
URL_RE = re.compile(
    r"(https?:\/\/[^\s]+)", # Busca cadenas que empiecen con http:// o
https:// seguidas de cualquier carácter no espacio
    re.IGNORECASE # Ignora mayúsculas/minúsculas al comparar (por
ejemplo, HTTP o http)
)

# --- Detecta nombres de archivos adjuntos en texto ---
ADJUNTO_RE = re.compile(
    r"(?<!\S)" # Asegura que antes no haya un carácter no-espacio (inicio
de palabra o después de un espacio)
    r"(?P<name>(?:[\\w\\-\\(\\)\\[\\]&]+(?:[ \\t]+[\\w\\-\\(\\)\\[\\]&]+)*)\\." # Captura
el nombre del archivo antes del punto
    r"(?P<ext>[A-Za-z0-9]{1,6}))" # Captura la extensión (1 a 6
letras/números)
    r"(?=[\\s\\)\\]\\.,;:!?])", # Asegura que después venga un espacio,
puntuación o final del texto
    re.UNICODE # Soporta caracteres Unicode en los nombres
)

# --- Conjuntos de extensiones de archivos ---
EXT_PELIGROSAS = { # Extensiones asociadas a ejecutables o scripts
peligrosos
    "exe", "bat", "cmd", "js", "vbs", "scr", "msi", "jar",
    "iso", "lnk", "ps1", "apk", "reg"
}

EXT_COMUNES = { # Extensiones típicas de documentos y archivos comprimidos
    "zip", "rar", "7z", "pdf", "doc", "docx", "docm",
    "xls", "xlsx", "xlsm", "ppt", "pptx", "pptm",
    "csv", "txt"
}

# --- Validación básica de correos electrónicos ---
```

```
EMAIL_RE = re.compile(
    r"^([^\s]+@[^\s]+\.[^\s]+)$", # Patrón: algo@algo.algo sin espacios
)
def es_email_valido(s: str) -> bool:
    """Devuelve True si el correo tiene un formato válido (mínimo
    algo@algo.algo)."""
    return bool(EMAIL_RE.match((s or "").strip())) # Limpia espacios y
    verifica coincidencia con el patrón
```

Se tienen las configuraciones de expresiones regulares en donde se tienen extensiones peligrosas o comunes para así detectar los archivos adjuntos peligrosos.

Archivo de Interfaz Gráfica

```
import re, threading # re: expresiones regulares; threading: ejecutar carga
del modelo en segundo plano
from pathlib import Path # Manejo de rutas de archivos de forma
multiplataforma
import tkinter as tk # Tkinter base
from tkinter import filedialog, messagebox # Diálogos para abrir archivos y
mostrar mensajes
from DeteccionDeSpam import EmailSpamClassifier # backend # Importa el
clasificador del backend
from Config_regex import URL_RE, ADJUNTO_RE, EXT_PELIGROSAS, EXT_COMUNES,
es_email_valido # Carga regex/sets/validador

# ===== Colores UI ===== # Paleta de colores para la interfaz
COLOR_BG = "#9ebbc0" # Color de fondo general
COLOR_INPUT = "#dfdede" # Color de fondo para entradas de texto
COLOR_TEXT = "#111827" # Color de texto principal

# ===== Ventana principal ===== # Configuración de la ventana raíz
root = tk.Tk() # Crea la ventana principal
root.title("Detector de Spam") # Título de la ventana
root.geometry("500x550") # Tamaño inicial de la ventana
root.configure(bg=COLOR_BG) # Aplica color de fondo
```

El resto del código es parte de la interfaz gráfica que no se considero necesario mostrar ya que solo se muestra cómo se creó el frame.

Pruebas del código

El código se inicia desde la clase de la Interfaz Gráfica:



The image shows a window titled "Detector de Spam" with a teal header bar. Inside, there are input fields for "Remitente:" (containing "juan.perez@empresa.com") and "Asunto:" (containing "Reunión semanal del equipo"). Below these is a large text area labeled "Contenido del mensaje:" containing a sample email body. At the bottom, there are three buttons: "Analizar", "Abrir archivo (.txt/.eml)", and "Limpiar". Below the buttons, it says "Modelo: Listo" with a checked checkbox. The bottom section, labeled "Resultados del análisis:", displays the extracted characteristics and classification results in a monospaced font.

Remitente:

Asunto:

Contenido del mensaje:

Hola equipo,

Les recuerdo que la reunión semanal será mañana a las 10:00 am en la sala de juntas.

Por favor, traigan los reportes actualizados.


Saludos,
Juan Pérez

Modelo: Listo ☒

Resultados del análisis:

```
=== Características extraídas ===
Remitente: juan.perez@empresa.com
Asunto: Reunión semanal del equipo
Enlaces (0):
Adjuntos mencionados: ninguno

=== Clasificación ===
Resultado: HAM
Confianza SPAM (mensaje): 0.58%
```


 **Detector de Spam** — □ ×

Remitente:

Asunto:

Contenido del mensaje:

Estimado cliente:

Detectamos actividad inusual en su cuenta.
Por favor, confirme sus datos accediendo al siguiente enlace:

<https://seguridadbancaria.com/validar-cuenta>

Si no actualiza su información en las próximas 24 horas, su cuenta podría ser suspendida.

Modelo: Listo ☒

Resultados del análisis:

Remitente: soporte@seguridadbancaria.com

Asunto: ¡Actualiza tu información urgente!

Enlaces (1):

- <https://seguridadbancaria.com/validar-cuenta>


Adjuntos mencionados (1):

- Instrucciones_actualizacion.pdf

=== Clasificación ===

Resultado: SPAM

Confianza SPAM (mensaje): 54.83%

 **Detector de Spam** — □ ×

Remitente:

premios@casino-oro.fun

Asunto:

¡100 tiradas gratis sin depósito!

Contenido del mensaje:

Gana dinero desde casa con nuestras tiradas gratis.
No necesitas tarjeta, solo haz clic en el enlace y comienza a ganar hoy.
<https://casino-oro.fun/bonus>
Adjunto: premios.zip

Analizar

Abrir archivo (.txt/.eml)

Limpiar

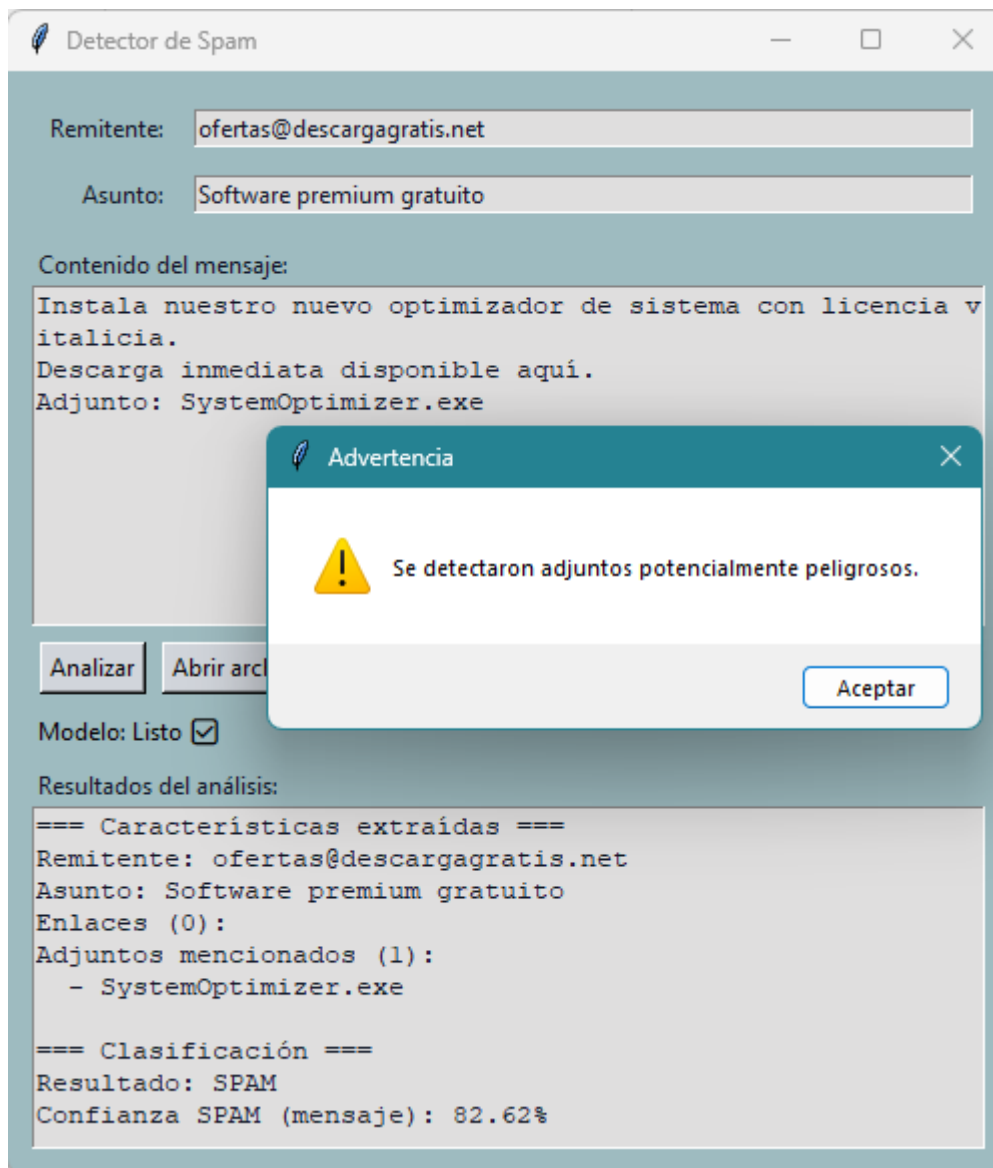
Modelo: Listo

☒


Resultados del análisis:

Remitente: premios@casino-oro.fun
Asunto: ¡100 tiradas gratis sin depósito!
Enlaces (1):
- <https://casino-oro.fun/bonus>
Adjuntos mencionados (1):
- premios.zip

=== Clasificación ===
Resultado: SPAM
Confianza SPAM (mensaje): 98.16%



En esta prueba se detectó un archivo adjunto con una extensión peligrosa por lo cual manda un mensaje de advertencia.

 Detector de Spam

Remitente:

profesor.lopez@universidad.edu

Asunto:

Entrega del proyecto final

Contenido del mensaje:

Recuerden que la entrega del proyecto será el lunes a las 18:00.
Suban el archivo en formato PDF al aula virtual.
Saludos cordiales.
Adjunto: guia_proyecto.docx

Analizar

Abrir archivo (.txt/.eml)

Limpiar

Modelo: Listo

☒

Resultados del análisis:

=== Características extraídas ===
Remitente: profesor.lopez@universidad.edu
Asunto: Entrega del proyecto final
Enlaces (0):
Adjuntos mencionados (1):
- guia_proyecto.docx

=== Clasificación ===
Resultado: HAM
Confianza SPAM (mensaje): 9.73%

 **Detector de Spam** — □ ×

Remitente:

Asunto:

Contenido del mensaje:

Hola!
Te mando las fotos del fin de semana en la sierra.
Nos vemos pronto.
Adjunto: viaje.zip

Analizar

Abrir archivo (.txt/.eml)


Limpiar

Modelo: Listo ☒

Resultados del análisis:

```
=== Características extraídas ===
Remitente: juan.perez@gmail.com
Asunto: Fotos del viaje
Enlaces (0):
Adjuntos mencionados (1):
- viaje.zip

=== Clasificación ===
Resultado: HAM
Confianza SPAM (mensaje): 2.16%
```

 **Detector de Spam** — □ ×

Remitente:

atencion.cliente@alerta-banco.com.co

Asunto:

Notificación Urgente: Su Tarjeta ha sido Bloqueada Temporalmente

Contenido del mensaje:

Hemos detectado un intento de acceso no autorizado. Para desbloquear su cuenta de inmediato y validar su identidad, haga clic en el siguiente enlace:

<http://verificacion-de-seguridad.es/usuario/login?id=2345>

Ignorar esta notificación resultará en la suspensión permanente. Por favor, confirme sus datos AQUI.

Analizar

Abrir archivo (.txt/.eml)

Limpiar

Modelo: Listo

☒

Resultados del análisis:

Asunto: Notificación Urgente: Su Tarjeta ha sido Bloqueada Temporalmente

Enlaces (1):

- <http://verificacion-de-seguridad.es/usuario/login?id=2345>

Adjuntos mencionados: ninguno

=== Clasificación ===

Resultado: SPAM

Confianza SPAM (mensaje): 59.07%

 **Detector de Spam** — □ ×

Remitente:

juan.perez@empresa.com

Asunto:

Reunión semanal del equipo

Contenido del mensaje:

Hola equipo,

Les recuerdo que la reunión semanal será mañana a las 10:00 am en la sala de juntas.
Por favor, traigan los reportes actualizados.

Saludos,
Juan Pérez

Analizar

Abrir archivo (.txt/.eml)

Limpiar

Modelo: Listo

☒

Resultados del análisis:

=== Características extraídas ===
Remitente: juan.perez@empresa.com
Asunto: Reunión semanal del equipo
Enlaces (0):
Adjuntos mencionados: ninguno

=== Clasificación ===
Resultado: HAM
Confianza SPAM (mensaje): 0.58%