Palomino Alejandra Victoria

**Informe Técnico del Proyecto Integrador - Análisis de Titulares Clickbait**

**Introducción**

El presente informe técnico del proyecto integrador, se realiza en el marco de las asignaturas “Técnicas de procesamiento digital de imágenes” y “Proyecto Integrador”, desarrollado en Python que permite analizar titulares de noticias para determinar si presentan características de clickbait (sensacionalistas) o son confiables. La aplicación combina técnicas de procesamiento de imágenes, OCR (Reconocimiento Óptico de Caracteres) y modelos de IA generativa, integradas en una interfaz gráfica de usuario. En términos generales, el sistema recibe como entrada un titular (ingresado manualmente, proveniente de una imagen o de un archivo CSV con múltiples titulares), aplica un preprocesamiento y extracción de texto (si es una imagen), luego realiza un análisis dual: un análisis elaborado manualmente, basado en reglas lógicas y patrones característicos del clickbait” y otro asistido por IA (modelo de IA de Google "Gemini"), y finalmente muestra el resultado comparativo al usuario. Los resultados de cada análisis se muestran en una tabla en la interfaz y se almacenan en archivos JSON para registro.

**Estructura y Módulos del Proyecto**

El proyecto se organiza en varios módulos (carpetas Python) separados, cada uno con funcionalidades específicas. Los principales módulos son: acciones/, analisis_manual/, gemini/, interfaz_grafica/ y procesamiento_de_imagenes/. A continuación, se explica el contenido de cada módulo y sus funciones más relevantes:

**Módulo acciones/**

Este módulo contiene funciones que conectan la interfaz gráfica con la lógica de análisis. En particular, gestiona las acciones del usuario, como cargar imágenes, ejecutar el análisis de un titular ingresado y limpiar la interfaz. Sus archivos principales son:

- **cargar_imagen.py**: Define la función **cargar_y_analizar_imagen(entrada_texto_usuario, tabla)**, la cual permite al usuario seleccionar un archivo de imagen desde el sistema de archivos y luego realiza tres pasos: (1) preprocesa la imagen para mejorar su calidad, (2) aplica OCR para extraer texto de la imagen, y (3) pasa el texto extraído al mecanismo de análisis

de titulares. La función utiliza filedialog.askopenfilename de Tkinter para abrir un cuadro de diálogo de selección de imagen (filtrando archivos .png, .jpg, etc.). Si el usuario cancela, la función termina sin cambios. Si se selecciona una imagen, se llama a **preparar_imagen(ruta)** (función del módulo de procesamiento de imágenes) para aplicar filtros de mejora (conversión a escala de grises, aumento de resolución, binarización, etc.), y se muestra la imagen resultante mediante **mostrar_imagen(imagen_procesada)** para que el usuario vea la imagen tras el preprocesamiento. Luego se extrae el texto de la imagen con **leer_texto(imagen_procesada)** (función OCR configurada con Tesseract). Si no se detecta texto, se emite una advertencia con messagebox.showwarning. Finalmente, invoca **ejecutar_analisis(...)** para analizar ese texto como si el usuario lo hubiese escrito manualmente. Toda la operación está envuelta en un bloque try/except que muestra un messagebox.showerror en caso de errores (por ejemplo, imagen corrupta o falla del OCR).

- **funciones.py**: Contiene las funciones generales para ejecutar el análisis de titulares y para limpiar la interfaz. Las principales son:
    - **ejecutar_analisis(entrada_texto_usuario, tabla)**: Gestiona el flujo completo de análisis cuando el usuario ingresa un titular de texto (en el campo de entrada de la GUI, Graphical User Interface) y presiona el botón de analizar. Primero, obtiene el texto escrito en el widget de texto (entrada_texto_usuario) y utiliza strip() para remover espacios o saltos sobrantes. Si el campo está vacío, muestra una advertencia y termina sin proceder. Si hay texto, llama a **analizar_titular_json(titular)** (importada del módulo analisis_manual) que realiza el análisis combinado manual + IA y devuelve un resultado en formato JSON (como cadena de texto). Esta cadena JSON luego se convierte a un diccionario Python para extraer los valores. En particular se obtienen: el titular analizado, el resultado de análisis manual (ej. "Clickbait" o "Confiable"), el resultado de análisis de IA (ej. "Dudoso Clickbait" o "Confiable") y si ambas evaluaciones coinciden o no. A partir de la clave "Coincidencia" se determina un *tag (etiqueta)* de formato para la fila de resultados en la tabla: si es "Verdadero" (ambas evaluaciones concuerdan) se usará un fondo verde; si es "Falso"

(evaluaciones discrepantes) fondo rojo; en otros casos, sin color especial. Luego la función inserta una nueva fila en la tabla de resultados de la interfaz con las cuatro columnas mencionadas (Titular, Análisis Manual, Análisis AI, Coincidencia), aplicando el color de fondo según el tag determinado. Después de mostrar el resultado en la tabla, la función limpia el campo de texto de entrada para que el usuario pueda ingresar un nuevo titular. A continuación, realiza el guardado: asegura que exista la carpeta resultados_json/ en el directorio del proyecto, define la ruta del archivo acumulativo resultados.json dentro de dicha carpeta, carga el contenido previo de ese archivo si existe, agrega el nuevo resultado (diccionario) a esa lista y vuelca de nuevo la lista completa al archivo JSON. De esta forma, se mantiene un historial acumulado de todos los titulares analizados. Finalmente, muestra un mensaje al usuario indicando que el análisis se realizó con éxito. En caso de que ocurra alguna excepción en cualquier punto del análisis, se notifica al usuario mediante un mensaje de error con el detalle del error.

- **ejecutar_titular(entrada_texto_usuario, tabla)**: El botón "Analizar" de la GUI utiliza ejecutar_analisis (vinculado a través de otra función), garantiza que cualquier llamada antigua a este nombre funcione igual.

- **limpiar_todo(entrada, tabla)**: Limpia la interfaz, tanto la entrada de texto como la tabla de resultados. Hace un borrado del contenido del widget de texto de entrada. Luego elimina todas las filas existentes en la tabla de resultados. Los try/except vacíos alrededor de estas operaciones ignoran errores inesperados. Esta función se invoca cuando el usuario pulsa el botón "Limpiar" en la interfaz, permitiendo reiniciar el formulario.

**Módulo analisis_manual/ – Análisis de titulares (manual y con IA)**

Este módulo contiene la lógica de análisis de los titulares de texto, incluyendo tanto el método basado en reglas manuales como la integración con la IA externa "Gemini". Sus archivos principales son:

- **evaluacion_manual_gemini.py**: Este archivo implementa dos funciones clave para analizar un titular:

- **analisis_manual(titular: str)**: Realiza un análisis determinístico del titular para decidir si es *clickbait*. Primero define listas de patrones típicos de titulares sensacionalistas: palabras_clickbait (frases como "último momento", "confirmado", "increíble", "te hará rico", "científicos dicen", "no podrás creer", "sorprendente", "viral", "secreto", "truco", "revelado", "exclusivo", "bomba", "escándalo", "shock", etc.) y numeros_clickbait (números comunes en títulos llamativos como "5", "10", "3", "7", "15" que suelen aparecer en listas o rankings). Luego inicializa un puntaje en 0. Suma puntos según varios criterios aplicados al titular de entrada:
  - Cuenta la cantidad de letras mayúsculas en el titular (caracteres que están en mayúscula). Si hay más de 5 mayúsculas en total, suma 1 punto (esto captura titulares escritos exageradamente en mayúsculas).
  - Cuenta la cantidad de signos de exclamación '!'. Cada exclamación suma 1 punto (titulares con muchos "!" suelen ser sensacionalistas).
  - Revisa si el titular contiene alguna de las palabras o frases de la lista palabras_clickbait (haciendo comparación en minúsculas para evitar case-sensitive). Por cada ocurrencia encontrada, suma 2 puntos (dando mayor peso a palabras clave claramente sensacionalistas).
  - Revisa si el titular contiene números "llamativos" de la lista numeros_clickbait. Por cada coincidencia, suma 1 punto.
- Tras estas evaluaciones, compara el puntaje total con un umbral: si el puntaje es mayor a 3, se considera el titular como "Clickbait"; si es 3 o menos, se clasifica como "Confiable". (Es decir, requiere al menos 4 puntos para etiquetar como clickbait). Se devuelve una tupla **(resultado_manual, puntaje)** con la categoría asignada y el puntaje calculado.
- **analizar_titular_json(titular: str)**: Esta función integra el análisis manual anterior con un análisis de IA (Gemini), devolviendo el resultado consolidado en formato JSON (como cadena). Su objetivo es tomar el titular, obtener tanto la evaluación manual como la de la IA, compararlas y empaquetarlas en un JSON. El flujo es el siguiente: primero verifica que el titular no esté vacío; si lo está, retorna un

mensaje de error en texto (ej. "Error: No se ingresó ningún titular."). Luego verifica que exista la clave de API para Gemini (importada como API_KEY_GEMINI desde gemini/keys/api_key.py); si no hay clave, retorna "Error: No se encontró la API key de Gemini.". Si todo responde correctamente, configura el cliente de la IA: utiliza la librería **Google Generative AI** para inicializar la API. Se elige el modelo llamado "gemini-2.5-flash". A continuación, la función construye un *prompt* en lenguaje natural que se enviará a la IA. El prompt le pide a la IA: *"Analizá el siguiente titular de noticia y clasificá su nivel de confiabilidad."* y provee criterios concretos:

- Si el titular tiene palabras exageradas, muchas mayúsculas o muchos signos de exclamación, la IA debe clasificarlo como **"Dudoso Clickbait"**.
- Si parece una noticia normal, objetiva y de fuente confiable, clasificar como **"Confiable"**.

  Además, se le instruye a la IA que responda estrictamente en formato JSON con la estructura: { "Evaluación Ai": "Confiable" o "Dudoso Clickbait" }. Finalmente se incluye el texto del titular en el prompt. Esto orienta al modelo generativo a devolver un JSON con la evaluación. Una vez preparado el prompt, se hace la llamada al modelo. Se espera que la respuesta del modelo contenga un texto en formato JSON con la evaluación. El código entonces procede a interpretar la respuesta:

- Si response.text no existe o está vacío, lanza un error indicando que *"Gemini no devolvió texto de respuesta."*.
- Luego el sistema intenta interpretar el texto generado por la IA como un JSON, es decir, convertirlo en una estructura de datos que el programa pueda manipular. analizar un texto y transformarlo en una estructura comprensible por el programa.
- Si valor sigue siendo None, el código aplica alternativas: usa una expresión regular para buscar un patrón. Si lo encuentra, extrae ese valor. Si aún no encuentra nada, como último recurso busca palabras clave dentro del texto devuelto: si en minúsculas encuentra "confiable", asigna valor = "Confiable";

si encuentra "dudoso" o "clickbait", asigna valor = "Dudoso Clickbait"; de lo contrario, asigna "No identificado". Estas comprobaciones aseguran que, incluso si la IA no respondió exactamente en el formato pedido, se pueda inferir la intención de su respuesta.

- En este punto, valor debería ser **"Confiable"** o **"Dudoso Clickbait"** (u otro texto identificable). Se guarda en resultado_ia.
- Paralelamente, antes de esto, la función ya ejecutó el análisis manual local: resultado_manual, _ = analisis_manual(titular). De modo que ahora tiene tanto resultado_manual como resultado_ia.
- Compara ambas evaluaciones para determinar coincidencia: si ambas consideran el titular confiable, o ambas lo consideran problemático (manual "Clickbait" y IA "Dudoso Clickbait"), entonces coincidencia = "Verdadero" (es decir, la IA concuerda con el análisis manual). En cualquier otro caso (por ejemplo, manual "Confiable" pero IA "Dudoso Clickbait", o viceversa), asigna coincidencia = "Falso".
- Luego arma un diccionario datos con las cuatro piezas de información: "Titular": titular, "Análisis Manual": resultado_manual, "Análisis Ai": resultado_ia, "Coincidencia": coincidencia.
- Convierte ese diccionario a una cadena JSON y la retorna. Si en cualquier punto del proceso con la IA ocurre una excepción, salta al except general y lanza un RuntimeError con un mensaje " ✕ Error al usar Gemini: {e}" que será capturado más arriba en la interfaz para informar al usuario.

Esta función combina los dos enfoques de análisis y formatea una respuesta unificada.

**cargar_archivo_csv.py**: Este archivo define la función **cargar_y_analizar_csv(entrada_texto_usuario, tabla)**, que permite al usuario procesar por partes varios titulares almacenados en un archivo CSV. Su funcionamiento es el siguiente: presenta un diálogo para elegir un archivo, filtrando solo *.csv. Si el usuario no selecciona nada, la función termina sin hacer

nada. Si hay un archivo seleccionado, usa pandas para leer el CSV. Luego verifica que en las columnas del DataFrame exista una columna llamada exactamente "titular"; de lo contrario, muestra : "Error", "El archivo debe tener una columna llamada 'titular' y termina el programa. Si la columna existe, extrae la serie de titulares, descarta valores NA, y convierte a lista de Python. Si la lista está vacía, avisa con ventana emergente que el archivo no tiene titulares y termina. Si hay titulares, se asegura de la existencia del directorio resultados_json. Define la ruta para resultados.json (archivo acumulativo), e intenta cargar los resultados previos de ese archivo (si está vacío o no existe, inicia una lista vacía). Luego itera sobre cada titula**r** en la lista: por cada uno, lo convierte a str por seguridad, y llama a **analizar_titular_json(titular)** para obtener el resultado en formato JSON (texto). Convierte ese JSON a un diccionario. Extrae los campos de interés (Titular, Análisis Manual, Análisis Ai, Coincidencia). Determina el color de la fila (color = ("coincide",) si coincidencia es "Verdadero", o ("no_coincide",) si no, similar a antes). Inserta la fila en la tabla con esos valores y color. Adicionalmente, agrega el diccionario de resultado a la lista todos_los_resultados. Al final del bucle, vuelca toda la lista actualizada a resultados.json. Finalmente muestra un messagebox.showinfo indicando cuántos titulares fueron analizados (por ejemplo, "Se analizaron 10 titulares."). Si ocurre algún error en el proceso, captura la excepción y muestra un mensaje de error : Error", "No se pudo procesar el archivo".

**Módulo gemini/ – Configuración de API de IA (Google)**

"Gemini" es el modelo de inteligencia artificial utilizado. En el proyecto, encontramos:

- **keys/api_key.py**: contiene la variable **API_KEY_GEMINI** con la clave de autenticación para el servicio de Google Generative AI. Dado que la clave es un dato sensible, se colocó en un archivo separado para no exponerlo en el código principal. La funcionalidad que tiene es almacenar la credencial de acceso a la API de IA.

**Módulo procesamiento_de_imagenes/ – Preprocesamiento de imágenes y OCR**

Este módulo agrupa las funciones relacionadas con el tratamiento de imágenes para mejorar la extracción de texto, así como la configuración del OCR Tesseract. Los archivos principales son:

- **arreglo_de_imagenes.py**: Implementa las funciones de preprocesamiento de la imagen antes de aplicar OCR. Usa la biblioteca OpenCV y Numpy para manipulación de matrices de imagen. Las funciones son:
    - **mostrar_imagen(imagen_preparada)**: Recibe una imagen y la muestra en una ventana aparte usando OpenCV. Para evitar imágenes grandes sobrepasen la pantalla, el código calcula una escala de reducción: limita el ancho a 800 px y alto a 600 px (tomando el mínimo de las escalas necesarias), luego redimensiona la imagen (cv2.resize) con esa escala usando interpolación de área (adecuada para reducir tamaño). Después, abre una ventana con la imagen. Esto pausa la ejecución hasta que el usuario pulse una tecla, y finalmente se cierra la ventana. Esta función se invoca después del preprocesamiento para permitir que el usuario visualice cómo quedó la imagen que será enviada al OCR. Además, cumple un rol importante en tareas de depuración (*debug*), ya que ayuda a identificar y corregir posibles errores en el procesamiento de la imagen.
    - **preparar_imagen(ruta_imagen)**: Ejecuta una cadena de transformaciones diseñadas para mejorar la imagen antes del reconocimiento óptico de caracteres.
        1. *Carga la imagen* desde el archivo usando cv2.imread(ruta_imagen). Si la imagen no se puede abrir (por ruta incorrecta u otro error), lanza un FileNotFoundError.
        2. *Convierte la imagen a escala de grises*: gray = cv2.cvtColor(imagen, cv2.COLOR_BGR2GRAY). Esto simplifica la información a una sola canal (intensidad) ya que el color no es necesario para OCR.
        3. *Aumenta la resolución (escalado):* para facilitar la lectura de caracteres pequeños, define un factor de escala (en el código, escala = 2.0, o sea duplica ancho y alto). Calcula las nuevas dimensiones y usa cv2.resize con

interpolación bicúbica (cv2.INTER_CUBIC) para agrandar la imagen a el doble de tamaño en cada dimensión. Esto puede ayudar a que letras pequeñas se representen con más pixeles y por tanto sean más reconocibles.

4. *Suaviza la imagen*: aplica un filtro Gaussiano cv2.GaussianBlur(gray, (3,3), 0) para reducir ruido. Un leve difuminado puede eliminar irregularidades de pixeles que podrían confundir al binarizar o al OCR.

5. *Binariza la imagen*: utiliza el método de umbralización de Otsu, muy común para separar texto del fondo. Otsu determina automáticamente un umbral óptimo para convertir la escala de grises en blanco/negro. La binarización de imágenes consiste precisamente en convertir una imagen a dos colores (blanco y negro) a partir de un umbral. Este paso es importante para aislar los caracteres del fondo.

6. *Afinar las letras:* esto tiende a *adelgazar* ligeramente los trazos y eliminar puntos de ruido pequeños. La idea es despegar letras que hayan quedado unidas o reducir imperfecciones tras la binarización. Repite la erosión 1 vez según el código.

7. Invertir colores si es necesario: calcula la media de intensidad de la imagen afinada. Si resulta que la imagen está oscura, significa que probablemente el texto quedó blanco sobre fondo negro (invertido), caso en el cual invierte la imagen para obtener texto negro sobre fondo blanco. Esto es importante porque Tesseract funciona mejor con texto oscuro sobre fondo claro (aunque puede leer ambos, esto elimina la posibilidad de que interprete fondo negro como texto).

8. Retorna la imagen procesada (afinada, ya binaria y optimizada).

Este *pipeline* de preprocesamiento está diseñado para maximizar la legibilidad del texto por parte del OCR. Es decir , incluye técnicas clásicas: escalado, filtrado, binarización (automática por Otsu), morfología e inversión condicionada.

- **tesseract_configuracion.py**: Configura y utiliza la herramienta **Tesseract OCR**. Contiene la función:
    - **leer_texto(imagen_preparada)**: Toma la imagen preprocesada y extrae el texto en ella usando **pytesseract**. Primero, si imagen_preparada es None, retorna cadena vacía inmediatamente (caso de error previo). Luego define la variable configuracion con una cadena de parámetros para Tesseract: "--oem 3 --psm 6 -l spa". Esto indica a Tesseract que use el OCR Engine Mode 3, que corresponde al motor basado en redes neuronales LSTM (es el más moderno y preciso en Tesseract 4+), y el Page Segmentation Mode 6, que asume un bloque de texto uniforme (parámetro adecuado cuando el texto puede tener varias líneas, pero está esencialmente en párrafos; a diferencia de 4 que asume columnas, o 7 que asumiría una sola línea). Además, se especifica -l spa para indicarle que use el *trained data* del idioma español, lo que mejora el reconocimiento de palabras en español. Con esta configuración ejecuta el OCR y devuelve el texto detectado. Seguidamente, realiza una limpieza de ese texto. Retorna el texto limpio, y enn caso de que pytesseract.image_to_string lance alguna excepción el código imprime un mensaje de error en consola [ERROR Tesseract]: {e} y retorna cadena vacía.

En contexto, leer_texto es utilizada después de preparar_imagen: el flujo completo para imágenes es *imagen original -> preparar_imagen -> mostrar_imagen (visualización) -> leer_texto -> resultado textual*.

**Módulo interfaz_grafica/ – Interfaz de Usuario (GUI)**

Este módulo contiene el código para la interfaz gráfica de usuario construida con Tkinter. El archivo principal es visualizacion_uno.py y además hay una subcarpeta resultados_json/ donde se guardan los resultados. A continuación, se describe la interfaz y su integración:

- **visualizacion_uno.py**: Al ejecutarlo, crea la ventana de la aplicación, todos los elementos gráficos (widgets) y enlaza los botones a las funciones anteriormente descritas. Los pasos y componentes principales son:

- **Inicialización de ventana**: Crea la ventana principal, fija el título de la ventana. Establece el fondo con color gris claro.
- **Título de la aplicación**: Coloca un texto grande en la parte superior, con el contenido *"Análisis de titulares Clickbait".*
- **Área de entrada de titular y botón Limpiar**: Crea un frame para agrupar la etiqueta y campo de texto de entrada junto con el botón de limpiar. Dentro de este frame:
  - Una etiqueta tk.Label indica *"Ingrese un titular:"* como texto instructivo.
  - Un botón *"Limpiar"* (tk.Button) a la derecha, que al pulsarse ejecuta la función limpiar_todo(entrada_texto_usuario, tabla) para borrar el campo y la tabla.
  - Un campo de texto multi-línea de altura 1 línea (se podría usar Entry de una línea, pero Text permite más flexibilidad) donde el usuario escribe o pega el titular a analizar.

- **Botones de acciones principales**: Debajo del frame anterior, en otro frame (frame_botones), se ubican tres botones importantes:
  - **Botón "Analizar"**: Inicia el análisis del titular actualmente escrito. En lugar de llamar directamente a ejecutar_analisis, el código define una función local llamada **analizar_y_guardar()** que encapsula la llamada a ejecutar_analisis y luego realiza acciones adicionales como guardar un JSON individual.
  - **Botón "Cargar archivo"**: Permite seleccionar un archivo CSV para análisis masivo. Al pulsarlo invoca cargar_y_analizar_csv(entrada_texto_usuario, tabla), la función del módulo analisis_manual descrita antes.

- **Botón "Cargar imagen"**: Abre un cuadro de diálogo para elegir una imagen y luego procesa su texto. Llama a cargar_y_analizar_imagen(entrada_texto_usuario, tabla) del módulo acciones. Etiquetado "Cargar imagen" con las mismas propiedades, colocado en columna 2.
- Los tres botones se distribuyen horizontalmente en el frame usando grid con separaciones, para que queden espaciados uniformemente.

- **Tabla de resultados**: Ocupa la parte inferior de la ventana. Implementada para mostrar columnas de resultados. Se coloca dentro de un frame panel_inferior con borde para separar visualmente. Se añaden también barras de desplazamiento vertical y horizontal vinculadas a la tabla.
- La tabla Treeview se configura con 4 columnas: *"Titular"*, *"Analisis Manual"*, *"Analisis Ai"* y *"Coincidencia"*
  - Se definen los encabezados visibles de cada columna.
  - Adicionalmente, el código del proyecto define estilos de colores para las filas con tags "coincide" y "no_coincide". Esto se hace creando estilos de tags. Dando al usuario una indicación visual inmediata de si la IA estuvo de acuerdo (verde) o no (rojo) con el análisis manual.
- **Lógica del botón "Analizar"** (analizar_y_guardar): En la interfaz, tras crear el botón Analizar, se configura su comando a la función local analizar_y_guardar. Esta función realiza los siguientes pasos:
  0. Cuenta cuántas filas había en la tabla antes del análisis.
  1. Llama a ejecutar_analisis(entrada_texto_usuario, tabla).
  2. Vuelve a contar filas (filas_despues). Si no hay más filas que antes, significa que no se realizó ningún análisis. En ese caso, la función termina retornando sin hacer nada más.

3. Si efectivamente se agregó una nueva fila, la localiza como la última fila insertada: ultima_fila.
4. Obtiene los valores de esa fila con tabla.item(ultima_fila, "values"), que devuelve la tupla (Titular, Análisis Manual, Análisis Ai, Coincidencia).
5. Si hay valores, construye un diccionario resultado con esas claves y valores.
6. Verifica que exista la carpeta resultados_json (si no, la crea, igual que en otras partes).
7. Genera un nombre de archivo único basado en la fecha-hora actual. Esto garantiza que cada análisis individual quede guardado en un archivo distinto, con marca temporal.
8. Guarda solo el último resultado individual en un JSON separado con indentación para legibilidad. Si ocurre algún error en este guardado, lo captura y muestra un messagebox.showerror("Error al guardar", "No se pudo guardar el resultado en JSON:\n{e}").
9. Arranca el loop de Tkinter con ventana_principal.mainloop()).

Mediante esta lógica, el sistema no solo mantiene un historial global en *resultados.json*, sino que también produce un archivo JSON independiente por cada análisis realizado manualmente. Esto podría ser útil para auditar casos individuales o para generar reportes puntuales.