

Extracción de texto de etiquetas del herbario de la UAS con técnicas de reconocimiento de caracteres.

Responsable: Dr. Inés Fernando Vega López.

Encargado: MC. Juan Augusto Campos Leal.

Participantes: José Julián López Rodríguez.

Índice

Capítulo 1: Introducción	1
Capítulo 2: Software necesario e instalación.	2
2.1 Sistema operativo y bibliotecas	2
2.2 Bibliotecas y versión de <i>Python</i>	2
2.3 Resumen	3
Capítulo 3: Proceso de recorte.	3
3.1 ¿Qué se hace en el proceso de recorte?	3
3.2 Ejecución del proceso de recorte	5
Capítulo 4: Preprocesamiento.	5
4.1 ¿Qué se hace en el preprocesamiento?	5
4.2 Filtros utilizados.	6
4.3 Ejecución del script para el preprocesamiento de imágenes..	7
Capítulo 5: Tesseract.	7
5.1 Información general sobre <i>Tesseract</i> y otros modelos de OCR.	7
5.2 Ejecución de <i>Tesseract</i>	7
Capítulo 6: Diccionarios.	8
6.1 Importancia de los diccionarios.	8
6.2 Obtención y generación de diccionarios	8
Capítulo 7: Proceso de comparación.	9
7.1 Distancia de Levenshtein y su implementación	9
7.2 Solución a problemas de acentos y letras mayúsculas.	9
7.3 Función para encontrar palabras similares.	10
7.4 Manejo de saltos de línea y guiones.	10
7.5 Evaluación de la confiabilidad de las extracciones.	11
Capítulo 8: Compacto.	12
Capítulo 9: Trabajo a futuro.	12

Capítulo 1: Digitalización del herbario de la UAS

El herbario de la Universidad Autónoma de Sinaloa (UAS) cuenta con un extenso registro de plantas disecadas. Lamentablemente, el acceso a estos especímenes es restringido, priorizando su conservación y limitando su disponibilidad al público en general, reservándose principalmente para investigadores. Ante esta situación, se ha propuesto la digitalización del herbario con el fin de democratizar el acceso a la información sobre especies y registros hallados en Sinaloa y otras regiones de México.

Para este propósito, se ha llevado a cabo la fotografía de cada una de las plantas o especímenes presentes en este herbario, totalizando más de 30,000 imágenes de alta resolución. En la Figura 1 se ejemplifica una de estas 30,000 imágenes.



Figura 1: Ejemplo de imagen del herbario.

Esto ha resultado en un amplio volumen de imágenes de alta resolución de la flora nativa de Sinaloa y México. No obstante, la digitalización real del herbario va más allá de las fotografías, ya que sería ineficiente que un usuario busque manualmente entre las imágenes hasta encontrar el espécimen deseado. Se requiere la extracción del texto de las etiquetas que proporcionan información sobre la especie, su ubicación y el investigador que la identificó, entre otros datos, para crear un herbario digital completo que no solo incluya la fotografía del espécimen, sino también su descripción, ubicación y autoría.

La extracción manual de información conlleva un gasto significativo de recurso humano. Por esta razón, el proyecto tiene como objetivo desarrollar una herramienta de inteligencia artificial basada en técnicas de visión por computadora. Esta herramienta estará dedicada a la detección automática de las etiquetas en las imágenes de los especímenes del herbario. Una vez localizada el área de la etiqueta de información, se aplicará un proceso de recorte y se utilizará una técnica de reconocimiento óptico de caracteres (OCR por su término en inglés), para extraer de manera automática el texto contenido en dicha etiqueta. Este enfoque permitirá agilizar y automatizar la extracción de información clave, reduciendo así la dependencia de la labor manual y optimizando el proceso de digitalización del herbario.

Capítulo 2: Software necesario e instalación.

2.1 Sistema operativo y bibliotecas.

Para este proyecto, se utilizó un equipo con *Ubuntu 22.04 LTS* y se instalaron las siguientes bibliotecas: *ImageMagick*, *libtesseract-dev* y *tesseract-ocr*. Estas herramientas pueden ser instaladas mediante los siguientes comandos en la terminal:

```

1 | sudo apt install libtesseract-dev=4.1.1-2.1build1
2 | sudo apt install tesseract-ocr=4.1.1
3 | sudo apt install imagemagick=6.9.11

```

2.2 Bibliotecas y versión de *Python*.

La versión de *Python* empleada fue la 3.8.0, en conjunto con las siguientes bibliotecas:

```

1 unicode=1.3.7
2 python-Levenshtein=0.23.0
3 opencv-python=4.8.1.78
4 numpy=1.24.3
5 pytesseract=0.3.10
6 pillow=8.3.2
7 pandas=1.3.3

```

Para instalar las bibliotecas se creó un ambiente de Anaconda en la versión 23.1.0. Dentro del ambiente virtual se utilizó el comando *pip* para instalar las bibliotecas.

2.3 Resumen.

Para la instalación completa de las herramientas necesarias, basta con ejecutar los siguientes comandos en la terminal de *Ubuntu* (dentro del ambiente virtual creado con Anaconda y asumiendo que *pip* está instalado):

```

1 sudo apt-get install libtesseract-dev=4.1.1-2.1build1
2 sudo apt-get install tesseract-ocr=4.1.1
3 sudo apt-get install imagemagick=6.9.11
4 pip install unidecode==1.3.7
5 pip install python-Levenshtein==0.23.0
6 pip install opencv-python==4.8.1.78
7 pip install numpy==1.24.3
8 pip install pytesseract==0.3.10
9 pip install pillow==8.3.2
10 pip install pandas==1.3.3

```

Es relevante señalar que estas versiones se utilizaron en el proyecto. No obstante, es probable que funcione con versiones superiores de estas bibliotecas y paquetes. Por ejemplo, se ha probado la ejecución en *Python 3.11.3*, aunque no es compatible con versiones anteriores a *Python 3.8*.

Capítulo 3: Proceso de recorte.

3.1 ¿Qué se hace en el proceso de recorte?

El objetivo de este proceso es tomar la imagen de un ejemplar y las coordenadas de la etiqueta en formato YOLO para recortar específicamente el área que contiene la etiqueta. Esto facilita la extracción del texto de dicha etiqueta. En la Figura 2 se muestra este proceso.

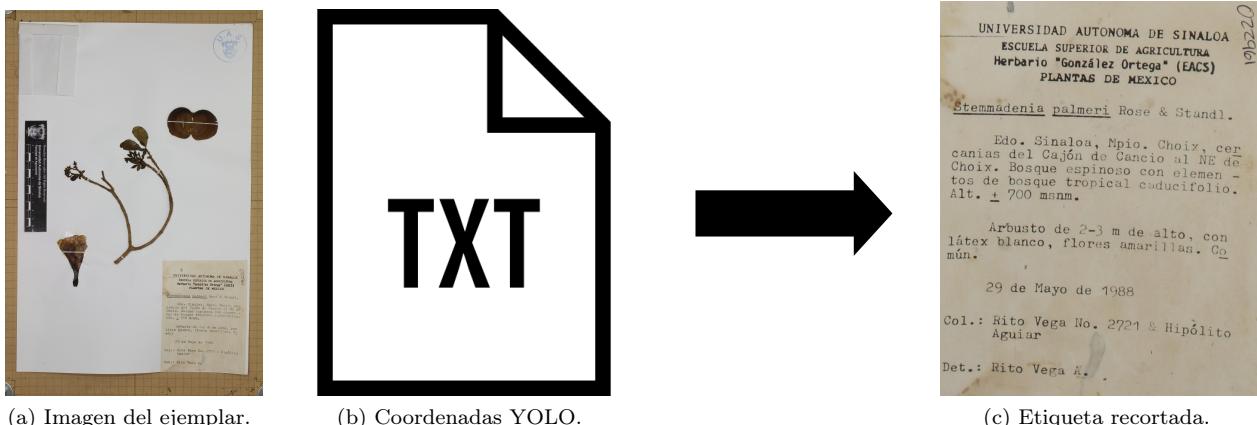


Figura 2: Automatización del recortado usando la imagen del ejemplar y las coordenadas YOLO.

Para realizar el recorte de las etiquetas, utilizamos el formato YOLO que viene representado por 5 valores

$$[A][X_{esc}][Y_{esc}][w_{esc}][h_{esc}]$$

donde cada número tiene un significado específico:

- $[A]$ representa el número de categoría del objeto detectado.
- $[X_{esc}]$ representa la coordenada x del centro del objeto detectado.
- $[Y_{esc}]$ representa la coordenada y del centro del objeto detectado.
- $[w_{esc}]$ representa el ancho del rectángulo que contiene al objeto detectado
- $[h_{esc}]$ representa la altura del rectángulo que contiene al objeto detectado

Las coordenadas proporcionadas (exceptuando la que representa el número de categoría del objeto detectado) se expresan como valores escalados entre 0 y 1. Dentro de este contexto, para las segundas y cuartas coordenadas, la unidad corresponde al ancho de la imagen, mientras que para las tercera y quintas coordenadas, la unidad representa la altura de la imagen. Para extraer las coordenadas a recortar, consideremos el siguiente ejemplo:

Ejemplo: Sea I una imagen de tamaño $W \times H$, y supongamos que las coordenadas de la etiqueta en formato YOLO son:

$$[A][X_{esc}][Y_{esc}][w_{esc}][h_{esc}]$$

El centro de la etiqueta sin escalar está dado por:

$$x_{centro} = X_{esc} \times W$$

$$y_{centro} = Y_{esc} \times H$$

El ancho y el alto de la etiqueta sin escalar son:

$$w = w_{esc} \times W$$

$$h = h_{esc} \times H$$

Finalmente, las coordenadas a recortar son desde las esquinas (X_1, Y_1) hasta (X_2, Y_2) , determinadas por:

$$X_1 = x_{centro} - \frac{w}{2}$$

$$Y_1 = y_{centro} - \frac{h}{2}$$

$$X_2 = x_{centro} + \frac{w}{2}$$

$$Y_2 = y_{centro} + \frac{h}{2}$$

El pseudocódigo para este procedimiento es el siguiente:

```

1 # Datos de la etiqueta en formato YOLO [A, X_esc, Y_esc, w_esc, h_esc]
2 [A, X_esc, Y_esc, w_esc, h_esc]
3
4 # Tamano de la imagen original
5 W, H = ObtenerTamanoImagen()
6
7 # Calcular las coordenadas no escaladas del centro de la etiqueta
8 x_centro = X_esc * W
9 y_centro = Y_esc * H
10
11 # Calcular el ancho y alto no escalados de la etiqueta
12 w = w_esc * W
13 h = h_esc * H
14
15 # Calcular las coordenadas del cuadro delimitador (bounding box)
16 X1 = x_centro - (w / 2)
17 Y1 = y_centro - (h / 2)
18 X2 = x_centro + (w / 2)
19 Y2 = y_centro + (h / 2)
20

```

```

21 # Las coordenadas X1, Y1 representan la esquina superior izquierda
22 # Las coordenadas X2, Y2 representan la esquina inferior derecha

```

Listing 1: Calculando Coordenadas del Cuadro Delimitador.

3.2 Ejecución del proceso de recorte.

Para ejecutar el proceso de recorte, sigue estos pasos:

1. Ve a la ruta del proceso de recorte, donde encontrarás dos carpetas: `ejemplares_etiquetados` y `etiquetas_recortadas`
2. En la carpeta `ejemplares_etiquetados`, guarda la imagen del ejemplar en formato JPG junto con su respectiva coordenada YOLO en un archivo de texto (.txt) que tengan el mismo nombre.
3. Ejecuta el script `recortador.py`.
4. Las etiquetas recortadas se guardarán en la carpeta `etiquetas_recortadas` en formato JPG.

Capítulo 4: Preprocesamiento.

4.1 ¿Qué se hace en el preprocesamiento?

En el preprocesamiento se aplica una serie de filtros con el objetivo de facilitar la tarea a *Tesseract* (el modelo de OCR utilizado en este proyecto) para la detección de caracteres. Es crucial tener en cuenta que la selección de filtros debe realizarse con precaución, ya que mientras un filtro de saturación puede mejorar la legibilidad en una imagen tenue, puede empeorar una etiqueta con manchas o ruido. Por ello, la elección de filtros debe ser cuidadosa, con la habilidad de adaptarse a cada caso específico.

En nuestro preprocesamiento, hemos identificado una serie de filtros que podemos aplicar para solucionar el problema. De la manera más automática posible. Eligiendo con cuidado que filtros pueden ayudar a la tarea. A continuación, presentamos ejemplos que ilustran su efecto en distintos tipos de imágenes.

En la Figura 3, observamos una etiqueta inicial tenue que, después del procesamiento, permite distinguir mejor los caracteres, lo que mejora significativamente el reconocimiento de *Tesseract*.

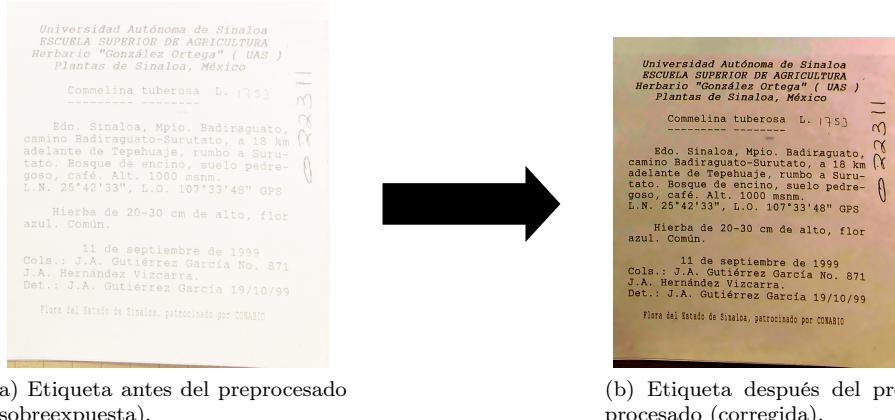
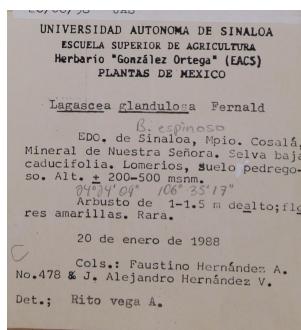
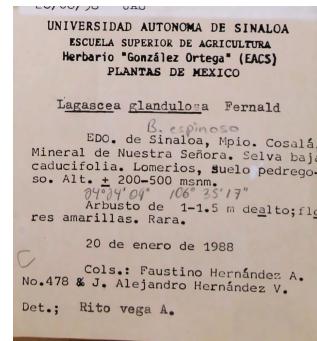


Figura 3: Mejora la legibilidad de la etiqueta sobreexpuesta con los filtros seleccionados.

En la Figura 4, notamos que la imagen inicial ya presenta una buena legibilidad de caracteres, por lo que el preprocesado no satura la imagen, ya que no es necesario.



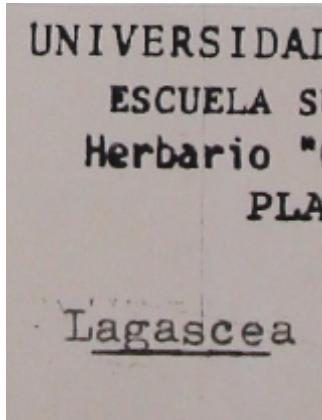
(a) Etiqueta antes del preprocesado (legible).



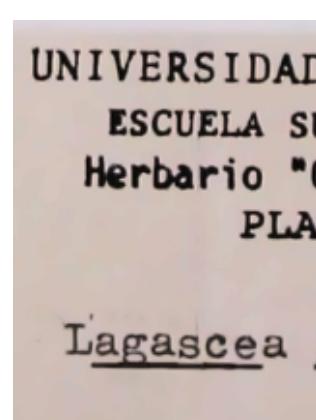
(b) Etiqueta después del preprocesado (no se sobresaturó).

Figura 4: Conservación de legibilidad en una etiqueta inicialmente legible.

Finalmente, en la Figura 5, mostramos cómo eliminamos manchas que suelen causar errores al aplicar OCR. Además, durante el preprocesado, también eliminamos el ruido y suavizamos la imagen para mejorar la calidad general del texto.



(a) Ampliación de una sección de la imagen antes del preprocesado con manchas.



(b) La misma sección después del preprocesado sin manchas.

Figura 5: Eliminación de manchas en una sección de la etiqueta con el preprocesado.

Esta etapa de preprocesamiento juega un papel fundamental en la mejora de la precisión del OCR al mejorar la calidad de las imágenes antes de que se realice la extracción de texto.

4.2 Filtros utilizados.

Durante el preprocesado, se emplearon los siguientes filtros:

- **Autogamma:** Este filtro ajusta automáticamente el brillo y contraste de una imagen mediante valores gamma, mejorando su apariencia sin perder detalles. Se utilizó la función correspondiente en *ImageMagick*. Para más información, se puede consultar: [Filtro autogamma de ImageMagick](#).
- **Normalización:** Ajusta el rango de brillo y contraste de una imagen para mejorar su interpretación. En este caso, se aplicó la función *normalize()* de *OpenCV*.
- **Redimensionamiento:** Para garantizar una adecuada resolución (más de 300 PPI) para el OCR, se redimensionó la imagen utilizando la biblioteca *Pillow* si la resolución era inferior a lo requerido.

- **Eliminación de ruido:** Se empleó la función *fastNlMeansDenoising* de *OpenCV* para eliminar puntos o manchas pequeños con alta intensidad en comparación con el resto de la imagen, suavizándola y mejorando su calidad.

4.3 Ejecución del script para el preprocesamiento de imágenes.

Para llevar a cabo el preprocesamiento de imágenes, sigue estos pasos:

1. Coloca las etiquetas a preprocesar en la carpeta `imagenesOriginals`.
2. Ejecuta el script `preprocesador.py`.
3. Las imágenes preprocesadas se guardarán en la carpeta `imagenesProcesadas`.

Capítulo 5: *Tesseract*.

5.1 Información general sobre *Tesseract* y otros modelos de OCR.

Tesseract es el modelo de OCR más popular en la actualidad, desarrollado por Google. Destaca por su versatilidad y alta capacidad de personalización, permitiendo su reentrenamiento para mejorar sus resultados. En la Figura 6 se ejemplifica la extracción de texto por parte de *Tesseract* a partir de una etiqueta.

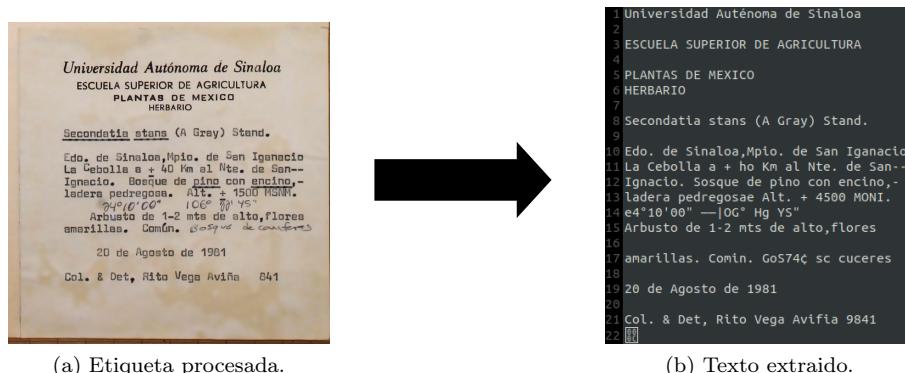


Figura 6: Extracción de texto por *Tesseract*.

Sin embargo, es importante notar que el OCR no es perfecto, como se observa en algunas palabras mal interpretadas, como «Autónoma» interpretada como «Auténoma», «bosque» como «sosque», «pedregosa» como «pedregosae», «común» como «comin», y «Aviña» como «Avifia». Además, las palabras escritas a mano presentan desafíos adicionales y no serán tratadas por el momento.

Es relevante mencionar que *Tesseract* no es el único modelo de OCR disponible. Otros dos modelos destacados en la actualidad son *PaddleOCR* y *EasyOCR*. Estos modelos también pueden ser reentrenados para mejorar sus resultados, al igual que *Tesseract*. La elección de *Tesseract* en este proyecto se basó en la amplia disponibilidad de información disponible en internet en comparación con los otros modelos mencionados.

5.2 Ejecución de *Tesseract*.

Para ejecutar *Tesseract*, sigue estos pasos:

1. Coloca las imágenes en la carpeta `imagenes`.
2. Ejecuta el script `tesseractUAS.py`.
3. El texto extraído se guardará en formato txt en la carpeta `textoExtraido`.

Capítulo 6: Diccionarios.

6.1 Importancia de los diccionarios.

En el capítulo anterior, se evidenció que el OCR no es perfecto y tiende a cometer errores al reconocer palabras, como en el caso de «bosque» interpretado como «sosque». Esta palabra, «sosque», no existe o al menos no se utiliza comúnmente en español. Esta situación nos brinda la oportunidad de corregir las palabras mal interpretadas por el OCR.

Para abordar esta problemática, se propone la creación de un repositorio de palabras, es decir, un diccionario, donde se almacenen palabras comunes. Este diccionario permitirá a la metodología, detectar una palabra como "sosque", buscarla en el diccionario y reemplazarla por la palabra más similar o probable. En el siguiente capítulo se abordará el proceso de comparación de palabras para llevar a cabo esta corrección.

Este capítulo se centrará en el desarrollo de este repositorio de palabras comunes, fundamental para mejorar la precisión y la exactitud en la interpretación del texto extraído por el OCR en este proyecto.

6.2 Obtención y generación de diccionarios

Para mejorar la precisión en la corrección de palabras mal interpretadas por el OCR, se trabajó en la creación de varios diccionarios específicos:

Apellidos americanos: Se recopilaron apellidos comunes en Estados Unidos a partir del sitio web [babynames.com](#), considerando la frecuencia de aparición.

Apellidos mexicanos: Se obtuvieron apellidos comunes en México a partir de la base de datos del Instituto Nacional de Estadística y Geografía (INEGI), disponible en la página [cuéntame](#), asegurándose de incluir acentos y corregir datos para una mejor precisión.

Especies de plantas: Se empleó una base de datos proporcionada por el Parque de Innovación Tecnológica (PIT) en colaboración con Comisión Nacional para el Conocimiento y Uso de la Biodiversidad (CONABIO) para incluir los nombres de especies de plantas más comunes.

Palabras comunes del español: Se utilizó un repositorio de GitHub [Diccionario-Español](#) que contenía palabras comunes en español, como artículos, pronombres y preposiciones.

Nombres comunes de México: Otra base de datos del INEGI ([INEGI](#)) se utilizó para recopilar nombres comunes en México, asegurando la inclusión de acentos.

Diccionario de números romanos: Se creó manualmente un diccionario de números romanos del 1 al 12 para la interpretación adecuada de fechas que empleaban este formato.

Diccionarios de números y palabras adicionales: Se agregaron diccionarios de números del 1 al 14,000, así como palabras específicas, como abreviaturas («Edo» para «Estado» o «mpio» para «municipio»), que no estaban presentes en las fuentes anteriores.

Cada diccionario se ajustó y se depuró según las necesidades del proyecto. Por ejemplo, en el caso de nombres y apellidos, se redujo la lista a los términos más comunes para evitar confusiones en la comparación de palabras. Además, se utilizaron herramientas como la biblioteca *Pandas* para eliminar duplicados y garantizar una mejor organización y eficiencia en los diccionarios generados.

Capítulo 7: Proceso de comparación.

7.1 Distancia de Levenshtein y su implementación

En esta sección, se emplea la distancia de Levenshtein para identificar palabras mal escritas o inexistentes detectadas por el OCR. La distancia de Levenshtein es definida como el número mínimo de operaciones requeridas para transformar una cadena de caracteres en otra.

Definición 1: La **distancia de Levenshtein**, también conocida como **distancia de edición** o **distancia entre palabras**, es el número mínimo de operaciones necesarias para transformar una cadena de caracteres en otra.

Ejemplo: Calcular la distancia de Levenshtein entre "bañeras" y "pradera":

La distancia de Levenshtein entre "bañeras" y "pradera" resulta en cuatro operaciones ya que:

1. bañeras → baderas (sustitución de 'ñ' por 'd')
2. baderas → paderas (sustitución de 'b' por 'p')
3. paderas → padera (eliminación de 's')
4. padera → pradera (inserción de 'r' entre 'p' y 'a')

El algoritmo se apoya en una matriz $(n+1) \times (m+1)$. Sean dos cadenas de caracteres, str1 y str2, de longitudes lenStr1 y lenStr2 respectivamente. La distancia de Levenshtein entre ellas se calcula mediante el siguiente pseudocódigo:

```

1 int DistanciaLevenshtein(char str1[1..lenStr1], char str2[1..lenStr2]) {
2     int d[lenStr1 + 1][lenStr2 + 1];
3     int i, j, costo;
4
5     for (i = 0; i <= lenStr1; i++)
6         d[i][0] = i;
7     for (j = 0; j <= lenStr2; j++)
8         d[0][j] = j;
9
10    for (i = 1; i <= lenStr1; i++) {
11        for (j = 1; j <= lenStr2; j++) {
12            if (str1[i - 1] == str2[j - 1])
13                costo = 0;
14            else
15                costo = 1;
16            d[i][j] = min(
17                d[i - 1][j] + 1,      # eliminacion
18                min(d[i][j - 1] + 1, # insercion
19                    d[i - 1][j - 1] + costo)); # sustitucion
20        }
21    }
22
23    return d[lenStr1][lenStr2];
24 }
```

Listing 2: Función para la distancia de Levenshtein.

Es relevante mencionar que, en lugar de implementar directamente el algoritmo de Levenshtein, se utilizó la biblioteca *Python-Levenshtein*.

7.2 Solución a problemas de acentos y letras mayúsculas.

Durante el uso de la biblioteca *Python-Levenshtein*, se identificaron inconvenientes con el OCR, que a veces detectaba palabras sin tilde cuando estas sí estaban presentes, y en otros casos colocaba tildes incorrectamente. Por ejemplo, detectaba la palabra «leon» en lugar de la palabra real «león». Esto plantea un problema importante:

La distancia de Levenshtein entre palabras como «leon» y «león» es de 1, ya que solo se requeriría la sustitución de la «o» sin tilde por una «o» con tilde. Sin embargo, la palabra «leen» también está a una distancia de 1 con la palabra «león». Este tipo de ambigüedad también se observó con las letras mayúsculas y minúsculas; por ejemplo, la distancia entre «Ave» y «ave» era de 1, a pesar de que solo existe una diferencia de mayúsculas y no de caracteres.

7.3 Función para encontrar palabras similares.

Para abordar las discrepancias con acentos y mayúsculas/minúsculas detectadas por el OCR, se implementó una solución utilizando la biblioteca *unidecode*, la cual elimina los acentos, y el método *lower()* para convertir todas las letras a minúsculas. Este enfoque permitió comparar las palabras sin considerar diferencias de acentos o casos.

A continuación se presenta la función utilizada para encontrar la palabra más similar a una palabra dada dentro de una lista:

```

1 def encontrar_palabra_similar(palabra, palabras):
2     distancia_minima = 1000000
3     distancia_minima2 = 1000000
4     palabras_similares = []
5     palabra_min = palabra.lower()
6
7     for p in palabras:
8         p_min = p.lower()
9         distancia = Levenshtein.distance(palabra_min, p_min, score_cutoff=distancia_minima)
10
11     if distancia <= distancia_minima:
12         distancia_minima = distancia
13         palabras_similares.append(p)
14     if distancia_minima == 0:
15         return p
16
17     palabra_min_sin_acentos = unidecode(palabra_min)
18     for palabra in palabras_similares:
19         palabra = palabra.lower()
20         palabra_sin_acentos = unidecode(palabra)
21         distancia = Levenshtein.distance(palabra_min_sin_acentos, palabra_sin_acentos)
22
23     if distancia < distancia_minima2:
24         distancia_minima2 = distancia
25         palabra_similar = palabra
26
27 return palabra_similar

```

Listing 3: Función para encontrar palabras similares.

Esta función, denominada `encontrar_palabra_similar()`, recibe una **palabra** y una lista de **palabras**. Utiliza la distancia de Levenshtein entre la palabra dada y cada palabra en la lista, considerando la eliminación de acentos y el cambio a minúsculas. Luego, devuelve la palabra más similar encontrada en la lista con respecto a la palabra proporcionada.

7.4 Manejo de saltos de línea y guiones.

Es importante destacar que en algunas ocasiones, para indicar un salto de línea cuando una palabra no cabe en una sola línea, los escritores de las etiquetas solían usar un guión, siguiendo las reglas gramaticales.

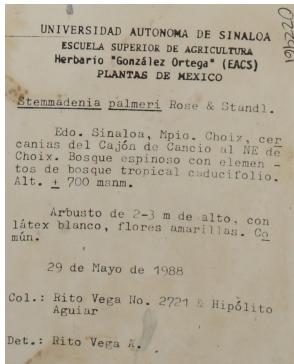


Figura 7: Etiqueta donde se muestra el uso del guión.

En la Figura 7 se muestra un ejemplo donde la palabra «elementos» se dividió en «elemen» y «tos» debido al salto de línea. Sin embargo, debido a como funciona el proceso de comparación, es necesario eliminar este salto de línea con guión, ya que solo se detecta palabras completas y no palabras interrumpidas por saltos de línea. Lamentablemente, no es posible corregir las palabras subrayadas antes del salto de línea, ya que el OCR no detecta el subrayado.

Este problema también afecta situaciones como la descripción de trayectos entre dos poblados, donde un salto de línea con guión interrumpe la continuidad de la palabra, por ejemplo:

Choix-
El Sauzal

Este texto se convierte en:

ChoixEl Sauzal

Se detecta «ChoixEl» como una palabra inexistente y la reemplaza por «Choix». En consecuencia, el texto final se transforma en:

Choix Sauzal

Esta peculiaridad en la escritura de las etiquetas implica la necesidad de implementar un tratamiento especial para unir palabras divididas por saltos de línea con guión para mantener la integridad de las palabras durante el proceso de comparación.

7.5 Evaluación de la confiabilidad de las extracciones.

Al finalizar cada extracción, se calcula la distancia de Levenshtein para cada palabra y se suma para obtener un promedio. Este promedio se utiliza para evaluar la confiabilidad de la extracción. Las extracciones con un promedio bajo de distancia de Levenshtein se consideran más confiables, indicando una menor cantidad de errores en la transcripción.

En el caso de que el promedio sea cero, existen dos posibilidades:

1. El OCR realizó una extracción perfecta, sin cometer errores en ninguna palabra.
2. El texto extraído está vacío, es decir, el OCR no detectó ninguna palabra en la imagen.

El resultado de este proceso se registra en un archivo de texto donde se presentan las extracciones consideradas más confiables y aquellas que se identifican como menos confiables, según el promedio de la distancia de Levenshtein.

Capítulo 8: Compacto.

La carpeta **compacto** simplifica y automatiza los procesos anteriores, permitiendo ejecutar cada uno de ellos de manera rápida y sencilla.

Ejecución:

Para extraer el texto de las imágenes de los ejemplares junto con las coordenadas de la etiqueta, sigue estos pasos:

1. Almacena las imágenes en formato JPG.
2. Asegúrate de tener el script `extractor.py` en tu sistema.
3. Ejecuta el script `extractor.py`, proporcionando las imágenes como entrada.
4. Verifica la existencia de la carpeta `texto_extraido`.
5. Busca el archivo `Promedio_de_distancias.txt`, generado por el script.

Una vez ejecutado el script, el archivo `Promedio_de_distancias.txt` contendrá el promedio de las distancias de Levenshtein de cada uno de los txt.

Capítulo 9: Trabajo a futuro.

Al próximo practicante o brigadista, se recomienda lo siguiente:

1. Es crucial investigar y mejorar los filtros utilizados. Los filtros desempeñan un papel fundamental en el proceso, ya que ayudan al modelo de OCR a distinguir las letras del fondo. Algunas imágenes pueden contener manchas, ser muy brillantes, tener un contraste tenue o incluso estar borrosas. Mejorar el preprocesamiento de las imágenes es esencial para que el OCR logre una precisión óptima.
2. Se sugiere explorar y comparar otros modelos de OCR disponibles además de *Tesseract*. Tanto Paddle OCR como Easy OCR ofrecen la posibilidad de reentrenar modelos. Esto resulta fundamental por varias razones:
 - Mejorar la detección de los símbolos ±.
 - Detectar adecuadamente el subrayado en las palabras.
 - Obtener resultados más precisos y mejorados en general.
3. Abordar el problema de saltos de línea con guión, ya que usualmente afecta la integridad de dos palabras ajenas durante el proceso de comparación.

Explorar y experimentar con diferentes modelos de OCR, contribuirá significativamente a mejorar la precisión y la capacidad de detección del sistema.