import spacy

import pandas as pd

import requests

from bs4 import BeautifulSoup

import re

import os

# Load the spaCy model

nlp = spacy.load("en\_core\_web\_sm")

def get\_text(input\_source):

    """Fetch text from different sources (direct text, URL, or file)"""

    # Case 1: Input is a URL

    if input\_source.startswith('http://') or input\_source.startswith('https://'):

        try:

            print(f"Fetching content from URL: {input\_source}")

            response = requests.get(input\_source, timeout=10)

            soup = BeautifulSoup(response.content, 'html.parser')

            # Remove script and style elements

            for script in soup(['script', 'style']):

                script.extract()

            text = soup.get\_text()

            # Clean excess whitespace

            text = re.sub(r'\s+', ' ', text).strip()

            print(f"Retrieved {len(text)} characters from URL")

            return text

        except Exception as e:

            print(f"Error fetching URL content: {e}")

            return None

    # Case 2: Input is a file path

    elif os.path.isfile(input\_source):

        try:

            print(f"Reading file: {input\_source}")

            with open(input\_source, 'r', encoding='utf-8') as f:

                content = f.read()

            print(f"Read {len(content)} characters from file")

            return content

        except Exception as e:

            print(f"Error reading file: {e}")

            return None

    # Case 3: Input is direct text

    else:

        print("Processing direct text input")

        return input\_source

def preprocess\_text\_to\_table(input\_source, selected\_features):

    """Process text and create feature table with selected annotations"""

    # Get text from various sources

    text = get\_text(input\_source)

    if not text:

        print("No valid text input received")

        return pd.DataFrame()

    # Process text with spaCy

    print("Processing text with spaCy...")

    doc = nlp(text)

    print(f"Processed {len(doc)} tokens")

    # Feature mapping dictionary

    feature\_map = {

        "Token": lambda t: t.text,

        "Lemma": lambda t: t.lemma\_,

        "POS": lambda t: t.pos\_,

        "Tag": lambda t: t.tag\_,

        "Dependency": lambda t: t.dep\_,

        "Is Stopword": lambda t: t.is\_stop,

        "Entity": lambda t: t.ent\_type\_,

        "Shape": lambda t: t.shape\_,

        "Alpha": lambda t: t.is\_alpha

    }

    # Validate selected features

    valid\_features = [f for f in selected\_features if f in feature\_map]

    if not valid\_features:

        print("No valid features selected. Using default set.")

        valid\_features = ["Token", "Lemma", "POS"]

    data = []

    for token in doc:

        if token.is\_space:

            continue

        row = {}

        for feature in valid\_features:

            row[feature] = feature\_map[feature](token)

        data.append(row)

    return pd.DataFrame(data)

def display\_feature\_menu():

    """Display feature selection menu"""

    print("\nAvailable features:")

    features = [

        "1. Token (Original word)",

        "2. Lemma (Base form)",

        "3. POS (Part-of-speech)",

        "4. Dependency (Syntactic relationship)",

        "5. Is Stopword (Common word filter)",

        "6. Alpha (Alphabetic character check)"

    ]

    for feature in features:

        print(feature)

    selections = input("\nSelect features (comma separated numbers, 'a' for all): ")

    if selections.lower() == 'a':

        return ["Token", "Lemma", "POS", "Dependency", "Is Stopword",  "Alpha"]

    feature\_map = {

        '1': "Token",

        '2': "Lemma",

        '3': "POS",

        '4': "Dependency",

        '5': "Is Stopword",

        '6': "Alpha"

    }

    selected = []

    for num in selections.split(','):

        num = num.strip()

        if num in feature\_map:

            selected.append(feature\_map[num])

    return selected if selected else ["Token", "Lemma", "POS"]

def main():

    """Main interactive program"""

    print("="\*50)

    print("TEXT ANALYSIS TOOL")

    print("="\*50)

    while True:

        print("\nInput options:")

        print("1. Enter text directly")

        print("2. Enter URL")

        print("3. Enter file path")

        print("4. Exit")

        choice = input("\nSelect input type (1-4): ").strip()

        if choice == '4':

            print("Exiting program...")

            break

        if choice == '1':

            input\_source = input("\nEnter text: ").strip()

        elif choice == '2':

            input\_source = input("Enter URL: ").strip()

        elif choice == '3':

            input\_source = input("Enter file path: ").strip()

        else:

            print("Invalid choice. Please try again.")

            continue

        selected\_features = display\_feature\_menu()

        df = preprocess\_text\_to\_table(input\_source, selected\_features)

        if df.empty:

            print("\nNo data to display. Please try another input source.")

            continue

        # Display results

        print("\n" + "="\*50)

        print("PROCESSED TEXT ANALYSIS")

        print("="\*50)

        print(f"Showing {len(df)} tokens with {len(selected\_features)} features")

        print(df.to\_string(index=False, max\_rows=25))

        # Save to CSV option

        save\_csv = input("\nSave to CSV? (y/n): ").lower()

        if save\_csv == 'y':

            filename = input("Enter filename (default: output.csv): ").strip()

            filename = filename if filename else "output.csv"

            if not filename.endswith('.csv'):

                filename += '.csv'

            df.to\_csv(filename, index=False)

            print(f"Results saved to {filename}")

        # Continue option

        another = input("\nProcess another input? (y/n): ").lower()

        if another != 'y':

            print("Exiting program...")

            break

if \_\_name\_\_ == "\_\_main\_\_":

    main()

import cv2

import numpy as np

# Initialize 3D cube parameters

cube\_size = 100

cube\_vertices = np.array([[-1,-1,-1], [1,-1,-1], [1,1,-1], [-1,1,-1],

                          [-1,-1,1], [1,-1,1], [1,1,1], [-1,1,1]], dtype=np.float32) \* cube\_size

edges = [(0,1), (1,2), (2,3), (3,0), (4,5), (5,6), (6,7), (7,4), (0,4), (1,5), (2,6), (3,7)]

colors = [(0,0,255), (0,255,0), (255,0,0), (255,255,0), (255,0,255), (0,255,255)]

# Camera parameters

focal\_length = 500

angle\_x, angle\_y = 0, 0

scale = 1.0

last\_mouse = (0, 0)

dragging = False

double\_click\_time = 0

crop\_mode = False

crop\_start, crop\_end = (0,0), (0,0)

def project\_point(point):

    x, y, z = point

    # Apply rotation

    rot\_x = np.array([[1,0,0], [0,np.cos(angle\_x),-np.sin(angle\_x)], [0,np.sin(angle\_x),np.cos(angle\_x)]])

    rot\_y = np.array([[np.cos(angle\_y),0,np.sin(angle\_y)], [0,1,0], [-np.sin(angle\_y),0,np.cos(angle\_y)]])

    point = rot\_x @ rot\_y @ np.array([x,y,z])

    # Apply perspective projection

    z = point[2] + 3\*cube\_size

    x = int(point[0]\*focal\_length/z + 400)

    y = int(point[1]\*focal\_length/z + 300)

    return (x, y)

def draw\_cube(img):

    projected = [project\_point(v\*scale) for v in cube\_vertices]

    # Draw edges with different colors

    for i, edge in enumerate(edges):

        color = colors[i%len(colors)]

        cv2.line(img, projected[edge[0]], projected[edge[1]], color, 2)

    # Draw vertices

    for p in projected:

        cv2.circle(img, p, 5, (255,255,255), -1)

    # Draw crop rectangle if in crop mode

    if crop\_mode and crop\_end != (0,0):

        cv2.rectangle(img, crop\_start, crop\_end, (0,255,0), 2)

def mouse\_callback(event, x, y, flags, param):

    global angle\_x, angle\_y, scale, last\_mouse, dragging, double\_click\_time, crop\_mode, crop\_start, crop\_end

    current\_time = cv2.getTickCount()

    if event == cv2.EVENT\_LBUTTONDOWN:

        if current\_time - double\_click\_time < cv2.getTickFrequency()\*0.3:

            scale \*= 1.5  # Double click - scale up

        double\_click\_time = current\_time

        last\_mouse = (x, y)

        dragging = True

    elif event == cv2.EVENT\_LBUTTONUP:

        dragging = False

    elif event == cv2.EVENT\_MOUSEMOVE:

        if dragging:

            dx, dy = x - last\_mouse[0], y - last\_mouse[1]

            angle\_x += dy\*0.01

            angle\_y += dx\*0.01

            last\_mouse = (x, y)

        elif crop\_mode and flags & cv2.EVENT\_FLAG\_MBUTTON:

            crop\_end = (x, y)

    elif event == cv2.EVENT\_MBUTTONDOWN and crop\_mode:

        crop\_start = (x, y)

        crop\_end = (x, y)

    elif event == cv2.EVENT\_MBUTTONUP and crop\_mode:

        crop\_end = (x, y)

        # Perform crop (resize)

        w, h = abs(crop\_end[0]-crop\_start[0]), abs(crop\_end[1]-crop\_start[1])

        if w > 10 and h > 10:  # Minimum size

            scale \*= min(1.0, min(w, h) / 200)  # Scale based on crop size

        crop\_start, crop\_end = (0,0), (0,0)

    elif event == cv2.EVENT\_RBUTTONDOWN:

        angle\_x, angle\_y = 0, 0  # Reset rotation

        scale = 1.0  # Reset scale

    elif event == cv2.EVENT\_MOUSEWHEEL:

        scale \*= 1.1 if flags > 0 else 0.9  # Zoom

# Create window and set mouse callback

cv2.namedWindow("3D Cube")

cv2.setMouseCallback("3D Cube", mouse\_callback)

while True:

    img = np.zeros((600, 800, 3), dtype=np.uint8)

    draw\_cube(img)

    # Display instructions

    cv2.putText(img, "Left drag: Rotate", (20, 30), cv2.FONT\_HERSHEY\_SIMPLEX, 0.7, (200,200,100), 2)

    cv2.putText(img, "Double click: Scale up", (20, 60), cv2.FONT\_HERSHEY\_SIMPLEX, 0.7, (200,200,100), 2)

    cv2.putText(img, "Right click: Reset", (20, 90), cv2.FONT\_HERSHEY\_SIMPLEX, 0.7, (200,200,100), 2)

    cv2.putText(img, "Mouse wheel: Zoom", (20, 120), cv2.FONT\_HERSHEY\_SIMPLEX, 0.7, (200,200,100), 2)

    cv2.putText(img, "Press 'c': Toggle crop mode", (20, 150), cv2.FONT\_HERSHEY\_SIMPLEX, 0.7, (200,200,100), 2)

    # Show crop mode status

    mode\_status = f"Crop Mode: {'ON' if crop\_mode else 'OFF'}"

    cv2.putText(img, mode\_status, (20, 550), cv2.FONT\_HERSHEY\_SIMPLEX, 0.7, (0,255,0) if crop\_mode else (0,0,255), 2)

    cv2.imshow("3D Cube", img)

    key = cv2.waitKey(30) & 0xFF

    if key == 27: break  # ESC key

    elif key == ord('c'): crop\_mode = not crop\_mode

cv2.destroyAllWindows()