Amazon ML Challenge

Team Name: Radium

Team Members:

- Sudharsanan K [Team Leader]
- Sabesh Raaj S
- Yogasri S
- Rathish Manivannan

Dataset Download:

```
import sys
import pandas as pd
sys.path.append(r'./amazon_ml/student_resource 3/src')
from utils import download_images

if _name_ == '_main_':
    csv_file = './amazon_ml/student_resource 3/dataset/test.csv'
    df = pd.read_csv(csv_file)
    image_links = df['image_link'].tolist()

    download_folder = "./test"
    download_images(image_links, download_folder, allow_multiprocessing=True)
```

Pre processing the data:

```
from PIL import Image, ImageEnhance
import cv2
import os
import numpy as np
from multiprocessing import Pool, cpu_count

def enhance_contrast(image, factor=2):
    """Enhance contrast of a PIL image."""
    enhancer = ImageEnhance.Contrast(image)
    return enhancer.enhance(factor)

def noise_removal(image):
    """Remove noise from an image using dilation and erosion."""
    kernel = np.ones((1, 1), np.uint8)
    image = cv2.dilate(image, kernel, iterations=1)
    image = cv2.erode(image, kernel, iterations=1)
    return image

def process_single_image(args):
    """Process a single image, enhance contrast and remove noise."""
    input_path, output_path, factor = args

    try:
        # Open the image using PIL and enhance contrast
        image = Image.open(input_path)
        enhanced_image = enhance_contrast(image, factor)

        # Convert to OpenCV format (numpy array)
        cv2_image = cv2.cvtColor(np.array(enhanced_image), cv2.COLOR_RGB2BGR)

        # Remove noise
        final_image = noise_removal(cv2_image)

        # Save the final processed image
        cv2.imwrite(output_path, final_image)
        print(f"Processed {os.path.basename(input_path)}")
    except Exception as e:
        print(f"Error processing {input_path}: {e}")

def process_images_in_directory(input_dir, output_dir, factor=2):
    """Process all images in the input directory using multiprocessing."""
```

```
    # Create the output directory if it doesn't exist
    os.makedirs(output_dir, exist_ok=True)

    # Prepare the list of image paths to process
    image_paths = [
        (os.path.join(input_dir, filename), os.path.join(output_dir, filename), factor)
        for filename in os.listdir(input_dir)
        if os.path.isfile(os.path.join(input_dir, filename))
    ]

    # Determine the number of processes to use (equal to the number of CPU cores)
    num_workers = cpu_count()
    print(f"Using {num_workers} parallel workers.")

    # Use a Pool to process images in parallel
    with Pool(processes=num_workers) as pool:
        pool.map(process_single_image, image_paths)

# Example usage
if _name_ == "_main_":
    input_directory = "./test"  # Directory containing the input images
    output_directory = "./test_preprocessed"  # Directory where processed images will be saved

    # Process all images in the input directory and save them in the output directory
    process_images_in_directory(input_directory, output_directory)
```

Training Spacy (name entity recognition trainer):

```
import random

# Units for each quantity
units = {
    "width": ["cm", "ft", "in", "m", "mm", "yd"],
    "depth": ["cm", "ft", "in", "m", "mm", "yd"],
    "height": ["cm", "ft", "in", "m", "mm", "yd"],
    "item_weight": ["g", "kg", "µg", "mg", "oz", "lb", "t"],
    "maximum_weight_recommendation": ["g", "kg", "µg", "mg", "oz", "lb", "t"],
    "voltage": ["kV", "mV", "V"],
    "wattage": ["kW", "W"],
    "item_volume": ["cL", "cu ft", "cu in", "cup", "dL", "fl oz", "gal", "imp gal", "L", "µL", "mL", "pint", "qt"]
}

def generate_training_data(num_sentences=5000):
    data = []
    for _ in range(num_sentences):
        quantity = random.choice(list(units.keys()))
        unit = random.choice(units[quantity])
        number = random.randint(1, 1000)
        sentence = f"The {quantity} is {number} {unit}."

        # Find the start and end indices of the number and unit using regex
        import re
        match = re.search(rf"({number})\s+({unit})", sentence)
        if match:
            number_start, number_end = match.start(1), match.end(1)
            unit_start, unit_end = match.start(2), match.end(2)

            # Create the entity annotation
            entity = [(number_start, unit_end, "QUANTITY")]
            data.append({"text": sentence, "entities": entity})
        else:
            print(f"Skipping sentence due to no match: {sentence}")
    return data

# Generate training data and save it to a JSON file
training_data = generate_training_data(num_sentences=5000)
print(training_data)



import pandas as pd
import os
from tqdm import tqdm
import spacy
from spacy.tokens import DocBin
```

```
nlp = spacy.blank("en")

db = DocBin() # create a DocBin object

for item in tqdm(training_data):  # item is a dictionary
    text = item["text"]  # get the text
    annot = item["entities"]  # get the entities
    doc = nlp.make_doc(text)  # create doc object from text
    ents = []
    for start, end, label in annot:  # add character indexes
        span = doc.char_span(start, end, label=label, alignment_mode="contract")
        if span is None:
            print(f"Skipping entity in text: {text}")
        else:
            ents.append(span)
    doc.ents = ents  # label the text with the ents
    db.add(doc)

db.to_disk("./train.spacy") # save the docbin object


nlp = spacy.load("./output/model-best")
```

Name entity extraction and writing output to a csv file:

```
import os
import pandas as pd
import re
import easyocr
import spacy
import csv
import sys
import time

# Load spaCy NER model for Quantity recognition
nlp = spacy.load("./output/model-best")  # Custom-trained NER model

# Conversion dictionary for full-form units
conversion_dict = {
    'width': {'cm': 'centimetre', 'ft': 'foot', 'in': 'inch', 'm': 'metre', 'mm': 'millimetre', 'yd': 'yard'},
    'depth': {'cm': 'centimetre', 'ft': 'foot', 'in': 'inch', 'm': 'metre', 'mm': 'millimetre', 'yd': 'yard'},
    'height': {'cm': 'centimetre', 'ft': 'foot', 'in': 'inch', 'm': 'metre', 'mm': 'millimetre', 'yd': 'yard'},
    'item_weight': {'g': 'gram', 'kg': 'kilogram', 'µg': 'microgram', 'mg': 'milligram', 'oz': 'ounce', 'lb': 'pound', 't':
    'maximum_weight_recommendation': {'g': 'gram', 'kg': 'kilogram', 'µg': 'microgram', 'mg': 'milligram', 'oz': 'ounce', 'l
    'voltage': {'kv': 'kilovolt', 'mv': 'millivolt', 'v': 'volt'},
    'wattage': {'kw': 'kilowatt', 'w': 'watt'},
    'item_volume': {'cl': 'centilitre', 'cu ft': 'cubic foot', 'cu in': 'cubic inch', 'cup': 'cup', 'dl': 'decilitre', 'fl o
}

# Cleaner Principle - Units for extraction and processing
units = {
    "width": ["cm", "ft", "in", "m", "mm", "yd"],
    "depth": ["cm", "ft", "in", "m", "mm", "yd"],
    "height": ["cm", "ft", "in", "m", "mm", "yd"],
    "item_weight": ["g", "kg", "µg", "mg", "oz", "lb", "t"],
    "maximum_weight_recommendation": ["g", "kg", "µg", "mg", "oz", "lb", "t"],
    "voltage": ["kv", "mv", "v"],
    "wattage": ["kw", "w"],
    "item_volume": ["cl", "cu ft", "cu in", "cup", "dl", "fl oz", "gallon", "l", "ml", "oz", "pint", "qt"]
}
all_units = [unit.lower() for sublist in units.values() for unit in sublist]

# EasyOCR Reader
reader = easyocr.Reader(['en'])

# Function to extract image name
def get_image_name(image_link):
    return os.path.basename(image_link)

# Function to convert abbreviations to full forms
def convert_units(entity_name, text):
    if entity_name in conversion_dict:
        for abbrev, full_form in conversion_dict[entity_name].items():
            text = re.sub(rf"\b{abbrev}\b", full_form, text)
```

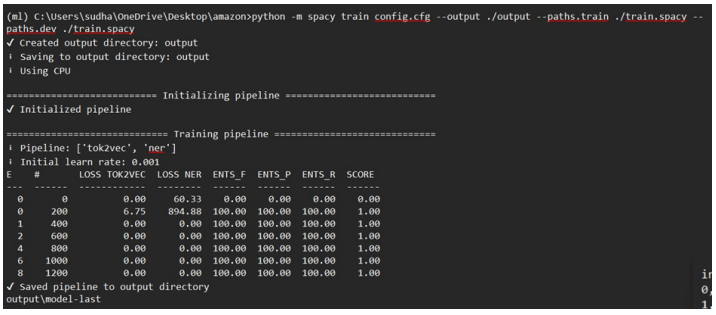
```
    return text

# Function to clean and extract measurements from text using regex
def extract_measurements(text):
    text = text.lower()  # Handle varying capitalizations
    pattern = r'(\d+(\.\d+)?)\s*({})'.format('|'.join(all_units))  # Pattern for numbers followed by units
    matches = re.findall(pattern, text)
    return [(match[0], match[2]) for match in matches]  # Return list of (value, unit) tuples

# Function to filter values based on the entity name
def get_value_for_entity(entity_name, measurements):
    if entity_name in units:
        valid_units = units[entity_name]
        for value, unit in measurements:
            if unit in valid_units:
                return f"{value} {unit}"
    return ""  # Return "0, empty string" if no valid measurement is found

# Function to process each image and extract the necessary quantity
def process_image(image_link, entity_name):
    image_name = get_image_name(image_link)
    image_path = os.path.join('./test_preprocessed', image_name)  # Assuming images are in 'images/' folder

    # Step 1: Extract text using EasyOCR
    result = reader.readtext(image_path, detail=0)
    extracted_text = " ".join(result)

    # Step 2: Perform NER with spaCy
    doc = nlp(extracted_text)
    quantities = [ent.text for ent in doc.ents if ent.label_ == "QUANTITY"]

    # Step 3: Clean the extracted text using the cleaner principle
    if quantities:
        measurements = extract_measurements(" ".join(quantities))
        # Step 4: Filter the extracted measurements for the given entity
        entity_value = get_value_for_entity(entity_name, measurements)

        if entity_value != "":
            # Step 5: Convert the abbreviation to full form
            entity_value = convert_units(entity_name, entity_value)
        return entity_value

    return ""

# Checkpoint logic
def save_checkpoint(last_index):
    with open('checkpoint.txt', 'w') as f:
        f.write(str(last_index))

def load_checkpoint():
    if os.path.exists('checkpoint.txt'):
        with open('checkpoint.txt', 'r') as f:
            return int(f.read().strip())
    return 0  # If no checkpoint file exists, start from the beginning

# Process all rows in CSV and collect results
def process_csv(input_file, output_file, batch_size=1000):

    df = pd.read_csv(input_file)

    # Load checkpoint (if exists)
    start_index = load_checkpoint()

    # Open the output CSV in append mode and skip writing header if file exists
    file_exists = os.path.isfile(output_file)
    with open(output_file, mode='a', newline='', encoding='utf-8') as outfile:
        fieldnames = ['index', 'prediction']
        writer = csv.DictWriter(outfile, fieldnames=fieldnames)

        # Write the header only if the file doesn't already exist
        if not file_exists:
            writer.writeheader()

        # Process rows starting from the checkpoint
        for index, row in df.iterrows():
            if index < start_index:
                continue  # Skip already processed rows
```

```
            image_link = row['image_link']
            entity_name = row['entity_name']
            index_value = row['index']

            # Extract and process image to get prediction
            prediction = process_image(image_link, entity_name)

            # Write result to the CSV file after processing each image
            writer.writerow({'index': index_value, 'prediction': prediction})

            print(f"Completed processing image {index_value}: {image_link}")

            # Flush the CSV file to ensure it's written to disk
            outfile.flush()

            # Save checkpoint every batch_size rows
            if (index + 1) % batch_size == 0:
                save_checkpoint(index + 1)
                print(f"Processed {index + 1} rows. Restarting... Last processed index saved.")
                restart_program()

    print(f"Processing completed. Predictions saved to {output_file}.")

# Function to restart the script
def restart_program():
    """
    This function schedules a restart of the current Python program.
    """
    time.sleep(30)
    python = sys.executable
    os.execl(python, python, *sys.argv)  # Restart the script using current arguments

# File paths
input_csv = './amazon_ml/student_resource 3/dataset/test.csv'  # Your input CSV with image links
output_csv = './new_test_out_final_final.csv'

# Run the processing with batch size of 500 rows
process_csv(input_csv, output_csv, batch_size=1000)
```

Final Results:

```
index,prediction
0,
1,42 centimetre
2,42 centimetre
3,42 centimetre
4,10.50 centimetre
5,10.50 centimetre
6,10.50 centimetre
7,
8,40 centimetre
9,40 centimetre
10,1 ton
11,6 centimetre
12,43 inch
13,43 inch
14,43 inch
15,
16,
17,
18,
19,240 volt
20,
21,
22,
23,032 centimetre
24,032 centimetre
```

Training output for spacy:



```
(ml) C:\Users\sudha\OneDrive\Desktop\amazon>python -m spacy train config.cfg --output ./output --paths.train ./train.spacy --
paths.dev ./train.spacy
✔ Created output directory: output
ℹ Saving to output directory: output
ℹ Using CPU

=========================== Initializing pipeline ===========================
✔ Initialized pipeline

============================= Training pipeline =============================
ℹ Pipeline: ['tok2vec', 'ner']
ℹ Initial learn rate: 0.001
E    #       LOSS TOK2VEC  LOSS NER  ENTS_F  ENTS_P  ENTS_R  SCORE
---  ------  ------------  --------  ------  ------  ------  ------
  0       0          0.00     60.33    0.00    0.00    0.00    0.00
  0     200          6.75    894.88  100.00  100.00  100.00    1.00
  1     400          0.00      0.00  100.00  100.00  100.00    1.00
  2     600          0.00      0.00  100.00  100.00  100.00    1.00
  4     800          0.00      0.00  100.00  100.00  100.00    1.00
  6    1000          0.00      0.00  100.00  100.00  100.00    1.00
  8    1200          0.00      0.00  100.00  100.00  100.00    1.00
✔ Saved pipeline to output directory
output\model-last
```

F1 Score : 0.328