CSE508 Information Retrieval
Winter 2024
Assignment-2 Report

Raj Gupta (2021410)

QUESTION: Perform the tasks specified below to make a Multimodal Retrieval System using Text as well as Images as the Input Data [100 Marks]

1. Image Feature Extraction [25]

A. Use basic image pre-processing techniques as altering contrast, resizing, geometrical orientation, random flips, brightness and exposure or any other relevant operation.

B. Use a pre-trained Convolutional Neural Network Architecture as ResNet, VGG16, Inception-v3, MobileNet (or any other CNN , preferably pre-trained on ImageNet Dataset), to extract relevant features from the images in the given training Set. Choose only one of the networks for your final pipeline.

C. Normalize the extracted features.

2. Text Feature Extraction [25]

A. Implement relevant pre-processing techniques as Lower-Casing, Tokenization, removing punctuations, Stop Word Removal, Stemming and Lemmatization on the given text reviews in the data

B. Calculate the Term Frequency-Inverse Document Frequency (TF-IDF) scores for the textual reviews.

Note: Please make sure to save your extracted features and the TF-IDF score using the pickle module so that you can run your code quickly in the demo

3. Image Retrieval and Text Retrieval [25]

A. For the input (image, review) pair, find the most similar images (preferably your top three) to your input based on extracted image features/embeddings using a similarity measure (cosine similarity) and a suitable data-structure.

B. For the input (image, review) pair, find the most similar reviews (preferably your top three) to your input review based on TF-IDF scores using a similarity measure (Cosine Similarity)

C. Save your results using Python's pickle module to save and load your results.

4. Combined Retrieval (Text and Image)

A. Get a composite similarity score (average) for the pairs generated in 3a) and 3b)

B. Rank the pairs based on the composite similarity score.

5. Results and Analysis

A. Present the top-ranked (image, review) pairs along with the cosine similarity scores.

B. Observe which out of the two retrieval techniques gives a better similarity score and argue the reason.

C. Discuss the challenges faced and potential improvements in the retrieval process.

SOLUTION:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import pickle
import requests
import nltk
import math
from collections import Counter
from nltk.stem import PorterStemmer
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from io import BytesIO
from sklearn.metrics.pairwise import cosine_similarity
from scipy.sparse import load_npz
from tensorflow.keras.applications.resnet50 import ResNet50, preprocess_input
from tensorflow.keras.preprocessing import image
from tensorflow.keras.models import Model
from sklearn.feature_extraction.text import TfidfVectorizer
```

```
[ ] nltk.download('punkt')
    nltk.download('stopwords')
    base_model = ResNet50(weights='imagenet')
    model = Model(inputs=base_model.input, outputs=base_model.layers[-2].output)
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

Importing and downloading the necessary libraries and modules for later use.

```
[ ] def extract_image_features(image_url):
        response = requests.get(image_url)
        img = image.load_img(BytesIO(response.content), target_size=(224, 224))
        img_array = image.img_to_array(img)
        expanded_img_array = np.expand_dims(img_array, axis=0)
        preprocessed_img = preprocess_input(expanded_img_array)
        features = model.predict(preprocessed_img)
        normalized_features = features / np.linalg.norm(features)
        return normalized_features.flatten()

    def preprocess_text(text):
        if isinstance(text, str):
            text = text.lower()
            tokens = word_tokenize(text)
            tokens = [word for word in tokens if word.isalpha()]  # Remove punctuation
            stop_words = set(stopwords.words('english'))
            tokens = [word for word in tokens if not word in stop_words]  # Remove stopwords
            porter = PorterStemmer()
            stemmed_tokens = [porter.stem(word) for word in tokens]
            return ' '.join(stemmed_tokens)
        else:
            return ""
```

The function ‘extract_image_features’ is used to extract features from the image. Image URL is given as the input, and the image is then loaded and converted into an array. After this, preprocessing steps are performed and features are extracted using the model (in this case, ResNet50). These features are then normalized.

The function ‘preprocess_text’ is used to perform preprocessing steps (like lowering the textcase, tokenization, punctuation removal, stopword removal and stemming). The function returns a string of these stemmed tokens as the output.

```
def tfidf(c):
    tokenized_docs = [doc.split() for doc in c]
    tf = [{word: doc.count(word) / len(doc) for word in set(doc)} for doc in tokenized_docs]
    df = defaultdict(int)
    num_docs = len(tokenized_docs)
    tfidf_matrix = []
    for doc in tf:
        tfidf_vector = {word: tf_value * idf[word] for word, tf_value in doc.items()}
        tfidf_matrix.append(tfidf_vector)
    return tfidf_matrix, idf
```

The function tfidf(c) computes the TF-IDF (Term Frequency-Inverse Document Frequency) matrix and IDF (Inverse Document Frequency) values for a given corpus c of documents. It tokenizes each document, calculates TF for each word, computes DF for each word, derives IDF values, and then computes TF-IDF scores for each word in each document, returning the TF-IDF matrix and IDF values.

```
[ ] df = pd.read_csv('A2_Data_final.csv')
    image_features = []
    for urls in df['Image']:
        urls = eval(urls)
        entry_features = np.mean([extract_image_features(url) for url in urls], axis=0)
        image_features.append(entry_features)

    processed_texts = [preprocess_text(text) for text in df['Review Text']]
    tfidf_matrix, tfidf_vectorizer = compute_tfidf(processed_texts)

    with open('image_features.pkl', 'wb') as f:
        pickle.dump(image_features, f)

    with open('tfidf_matrix.pkl', 'wb') as f:
        pickle.dump(tfidf_matrix, f)

    with open('tfidf_vectorizer.pkl', 'wb') as f:
        pickle.dump(tfidf_vectorizer, f)
```

The code reads a CSV file (named ‘A2_Data_final.csv’) containing image URLs and reviews, processes the images to extract features, and pre-processes the text. It then computes TF-IDF matrices for the text data and saves the image features (using pickle), TF-IDF matrix, and vectorizer into pickle files for later use.

```
[ ] def find_top_similarities(input_vector, feature_matrix, top_n=3):
        similarities = cosine_similarity(input_vector.reshape(1, -1), feature_matrix)
        top_indices = similarities.argsort()[0][-top_n-1:-1][::-1]
        return top_indices, similarities[0][top_indices]

    def retrieve_similar_items(input_index, feature_matrix, is_text=False, top_n=3):
        if is_text:
            input_vector = feature_matrix[input_index]
        else:
            input_vector = feature_matrix[input_index].reshape(1, -1)

        top_indices, top_similarities = find_top_similarities(input_vector, feature_matrix, top_n)
        return top_indices, top_similarities

    def combined_retrieval(image_input_index, text_input_index, image_features, tfidf_matrix, top_n=3):
        image_feature_matrix = np.array(image_features)

        image_similarities = cosine_similarity(image_feature_matrix[image_input_index].reshape(1, -1), image_feature_matrix).flatten()
        text_similarities = cosine_similarity(tfidf_matrix[text_input_index], tfidf_matrix).flatten()

        combined_scores = (image_similarities + text_similarities) / 2

        top_indices = np.argsort(combined_scores)[-top_n-1:-1][::-1]

        results = [(idx, {'image': image_similarities[idx], 'text': text_similarities[idx], 'average': combined_scores[idx]}) for idx in top_indices]

        return results
```

The ‘find_top_similarities()’ function computes the cosine similarity between an input vector and each row in the feature matrix. It then returns the indices and similarity scores of the top 3 most similar rows in the features matrix to the input vector.

The function ‘retrieve_similar_items()’ takes an input index and a feature matrix, with an optional flag for text data and another for specifying the number of similar items to retrieve (in this case, 3). It retrieves the input vector from the feature matrix and then finds the top similar items using the function and returns the indices and similarities of these top similar items.

The ‘combined_retrieval()’ function takes an image and text index along with their feature representations. It computes cosine similarities between the input image and all images, and the input text and all texts. These similarities are averaged to get combined scores. The function then selects the top similar pairs based on these scores and returns them along with similarity metrics for both image and text.

```
[ ] def load_data():
        df = pd.read_csv('A2_Data_final.csv')
        return df

    def load_features():
        with open('image_features.pkl', 'rb') as f:
            image_features = pickle.load(f)
        with open('tfidf_matrix.pkl', 'rb') as f:
            tfidf_matrix = pickle.load(f)
        with open('tfidf_vectorizer.pkl', 'rb') as f:
            tfidf_vectorizer = pickle.load(f)
        return image_features, tfidf_matrix, tfidf_vectorizer
```

The function ‘load_data()’ is used to read and return the given CSV file.

The function ‘load_features()’ is used to load the image_features.pkl, tfidf_matrix.pkl, and tfidf_vectorizer.pkl using the pickle module. These pickle files are later used to evaluate the input.

```
def main():
    df = load_data()
    image_features, tfidf_matrix, tfidf_vectorizer = load_features()

    # Take input from the user for image URL and review text
    image_url = input("Please enter the image URL: ")
    review_text = input("Please enter the review text: ")

    # Now you can use these variables in your code
    print("Image URL:", image_url)
    print("Review Text:", review_text)

    print("6. Sample Test Case:")
    print("a. Input:")
    print("Image and Text Query Input:")
    print("Image:")
    print(image_url)
    print("Review:", review_text)

    combined_results = combined_retrieval(0, 0, image_features, tfidf_matrix)
```

```
    print("b. Output:")
    print("--------------------------------------------------------------------------------")
    print("USING IMAGE RETRIEVAL")
    for i, (idx, result) in enumerate(combined_results[:3], start=1):
        print(f"Image URL: {df['Image'][idx]}")
        print(f"Review: {df['Review Text'][idx]}")
        print(f"Cosine similarity of images - {result['image']:.4f}")
        print(f"Cosine similarity of text - {result['text']:.4f}")
        print(f"Composite similarity score: {result['average']:.4f}")
        print()

    combined_results = combined_retrieval(0, 0, image_features, tfidf_matrix)
    text_results = retrieve_similar_items(0, tfidf_matrix, is_text=True)

    print("--------------------------------------------------------------------------------")
    print("USING TEXT RETRIEVAL")
    for i, (idx, similarity) in enumerate(zip(*text_results), start=1):
        print(f"Image URL: {df['Image'][idx]}")
        print(f"Review: {df['Review Text'][idx]}")
        print(f"Cosine similarity of images - {combined_results[i-1][1]['image']:.4f}")
        print(f"Cosine similarity of text - {similarity:.4f}")
        print(f"Composite similarity score: {combined_results[i-1][1]['average']:.4f}")
        print()
```

The ‘main()’ function loads the database and then loads the pickle files using the load_features() function. After this, user input for image URL and review text is taken. After this, using image retrieval, the top 3 images most similar to the input image are retrieved and printed (along with their corresponding image URL, review text, cosine similarity score of image and review text, and composite cosine similarity score). The similar is done for the text retrieval method, where the top 3 review texts most similar to the input review text are retrieved and printed (along with their corresponding image URL, review text, cosine similarity score of image and review text, and composite cosine similarity score).