Text Summarization Code Working

Extractive Method

Extractive text summarization is a natural language processing (NLP) technique used to create a concise summary of a longer text by selecting and extracting key sentences or phrases directly from the original content. Instead of generating new sentences or content like abstractive summarization, extractive summarization relies on identifying and ranking the most informative and relevant parts of the source text to present a coherent and informative summary.

import nltk

from nltk.corpus import stopwords

from nltk.cluster.util import cosine\_distance

import numpy as np

import networkx as nx

def read\_article(file\_name):

file = open(file\_name, "r")

filedata = file.readlines()

article = filedata[0].split(". ")

sentences = []

for sentence in article:

print(sentence)

sentences.append(sentence.replace("[^a-zA-Z]", " ").split(" "))

sentences.pop()

return sentences

The function called read\_article(file\_name) that reads the contents of a file, assumes the first line contains an article, and processes it into individual sentences. It attempts to remove non-alphabetical characters and returns the sentences as lists of words.

def sentence\_similarity(sent1, sent2, stopwords=None):

if stopwords is None:

stopwords = []

sent1 = [w.lower() for w in sent1]

sent2 = [w.lower() for w in sent2]

all\_words = list(set(sent1 + sent2))

vector1 = [0] \* len(all\_words)

vector2 = [0] \* len(all\_words)

# build the vector for the first sentence

for w in sent1:

if w in stopwords:

continue

vector1[all\_words.index(w)] += 1

# build the vector for the second sentence

for w in sent2:

if w in stopwords:

continue

vector2[all\_words.index(w)] += 1

return 1 - cosine\_distance(vector1, vector2)

It calculates the similarity between two sentences based on the words they share while ignoring stopwords and word order. It does this by creating word vectors and using cosine similarity.

def build\_similarity\_matrix(sentences, stop\_words):

# Create an empty similarity matrix

similarity\_matrix = np.zeros((len(sentences), len(sentences)))

for idx1 in range(len(sentences)):

for idx2 in range(len(sentences)):

if idx1 == idx2: #ignore if both are same sentences

continue

similarity\_matrix[idx1][idx2] = sentence\_similarity(sentences[idx1], sentences[idx2], stop\_words)

return similarity\_matrix

This function takes a list of sentences and a list of stopwords as input and computes a similarity matrix where each element represents the similarity between two sentences based on the sentence\_similarity function. The resulting matrix can be used for various text analysis tasks, such as clustering or summarization.

def generate\_summary\_to\_file(input\_file\_name, output\_file\_name, top\_n=5):

nltk.download("stopwords")

stop\_words = stopwords.words('english')

summarize\_text = []

# Step 1 - Read text and split it

sentences = read\_article(input\_file\_name)

# Step 2 - Generate Similarity Matrix across sentences

sentence\_similarity\_matrix = build\_similarity\_matrix(sentences, stop\_words)

# Step 3 - Rank sentences in similarity matrix

sentence\_similarity\_graph = nx.from\_numpy\_array(sentence\_similarity\_matrix)

scores = nx.pagerank(sentence\_similarity\_graph)

# Step 4 - Sort the rank and pick top sentences

ranked\_sentences = sorted(((scores[i], s) for i, s in enumerate(sentences)), reverse=True)

for i in range(top\_n):

summarize\_text.append(" ".join(ranked\_sentences[i][1]))

# Step 5 - Write the summarized text to the output file

with open(output\_file\_name, "w") as output\_file:

output\_file.write(". ".join(summarize\_text))

This function performs extractive text summarization by calculating the importance of sentences using sentence similarity and PageRank. It then selects the top-ranked sentences and saves them as a summary in the specified output file. The number of sentences in the summary can be controlled by the top\_n parameter.

# Specify the input and output file names

input\_file = "Text.txt"

output\_file = "summary.txt"

# Generate and save the summary to the output file

generate\_summary\_to\_file(input\_file, output\_file, 2)

Abstractive Method

It’s a technique that aims to generate a concise summary of a longer text by interpreting and rephrasing the content in a human-readable and coherent manner. Unlike extractive summarization, which selects and combines existing sentences or phrases from the source text, abstractive summarization involves creating entirely new sentences that capture the essential information and meaning of the original text.

!pip3 install torch torchvision torchaudio

!pip install transformers

!pip install sentencepiece

from transformers import PegasusForConditionalGeneration, PegasusTokenizer

import torch

Pegasus is a pre-trained model designed for abstractive text summarization.

import torch: This line imports the PyTorch library, which is required for using deep learning models, including Pegasus.

model\_name = 'google/pegasus-xsum'

torch\_device = 'cuda' if torch.cuda.is\_available() else 'cpu'

By setting torch\_device based on GPU availability, we ensure that the code will work efficiently on different hardware configurations, making it more versatile and potentially faster when using GPU acceleration.

tokenizer = PegasusTokenizer.from\_pretrained(model\_name)

It's used to convert text into a format that can be understood by the model, which typically involves splitting the text into smaller units like words or subwords.

from\_pretrained(model\_name): This method initializes the tokenizer using a pre-trained Pegasus model specified by the model\_name variable. The model\_name variable, in this case, is set to 'google/pegasus-xsum', which refers to a pre-trained Pegasus model fine-tuned on the XSum dataset.

model = PegasusForConditionalGeneration.from\_pretrained(model\_name).to(torch\_device)

PegasusForConditionalGeneration: This is a class provided by the Transformers library specifically designed for fine-tuned Pegasus models used in conditional text generation tasks, such as text summarization. It's capable of taking an input text and generating a corresponding summary.

.from\_pretrained(model\_name): This method loads a pre-trained Pegasus model specified by the model\_name variable. In the code, model\_name is set to 'google/pegasus-xsum', which corresponds to a pre-trained Pegasus model fine-tuned on the XSum dataset.

.to(torch\_device): This part of the code transfers the loaded model to the computing device specified by the torch\_device variable. If torch\_device is set to 'cuda', it means the model will run on a CUDA-compatible GPU if available, and if torch\_device is set to 'cpu', it will run on the CPU.

src\_text = [

"""In an attempt to build an AI-ready workforce, Microsoft announced Intelligent Cloud Hub which has been launched to empower the next generation of students with AI-ready skills.

Envisioned as a three-year collaborative program, Intelligent Cloud Hub will support around 100 institutions with AI infrastructure, course content and curriculum, developer support,

development tools and give students access to cloud and AI services. As part of the program, the Redmond giant which wants to expand its reach and is planning to build a strong developer

ecosystem in India with the program will set up the core AI infrastructure and IoT Hub for the selected campuses. The company will provide AI development tools and Azure AI services

such as Microsoft Cognitive Services, Bot Services and Azure Machine Learning.According to Manish Prakash, Country General Manager-PS, Health and Education, Microsoft India, said,

"With AI being the defining technology of our time, it is transforming lives and industry and the jobs of tomorrow will require a different skillset. This will require more

collaborations and training and working with AI. That’s why it has become more critical than ever for educational institutions to integrate new cloud and AI technologies. The program

is an attempt to ramp up the institutional set-up and build capabilities among the educators to educate the workforce of tomorrow." The program aims to build up the cognitive skills

and in-depth understanding of developing intelligent cloud connected solutions for applications across industry. Earlier in April this year, the company announced Microsoft Professional

Program In AI as a learning track open to the public. The program was developed to provide job ready skills to programmers who wanted to hone their skills in AI and data science with a

series of online courses which featured hands-on labs and expert instructors as well. This program also included developer-focused AI school that provided a bunch of assets to help build

AI skills."""

]

batch = tokenizer.prepare\_seq2seq\_batch(src\_text, truncation=True, padding='longest',return\_tensors='pt')

translated = model.generate(\*\*batch)

tgt\_text = tokenizer.batch\_decode(translated, skip\_special\_tokens=True)

src\_text: This is a list containing a single string that represents the source text for which you want to generate a summary. The source text is a news article or document in this case.

batch: This variable is created by preparing a sequence-to-sequence batch using the Pegasus tokenizer. It takes the src\_text, truncates it (if needed), pads it to the longest sequence, and returns the batch in PyTorch format ('pt').

translated: This line uses the pre-trained Pegasus model (model) to generate a summary for the source text. It passes the prepared batch as arguments to the model.generate() method. The generate method returns the generated summary.

tgt\_text: This variable contains the decoded summary generated by the model. It decodes the summary text and skips special tokens, such as [PAD] and [CLS], to produce the final readable summary.

print(tgt\_text)