Machine Learning - CS6301

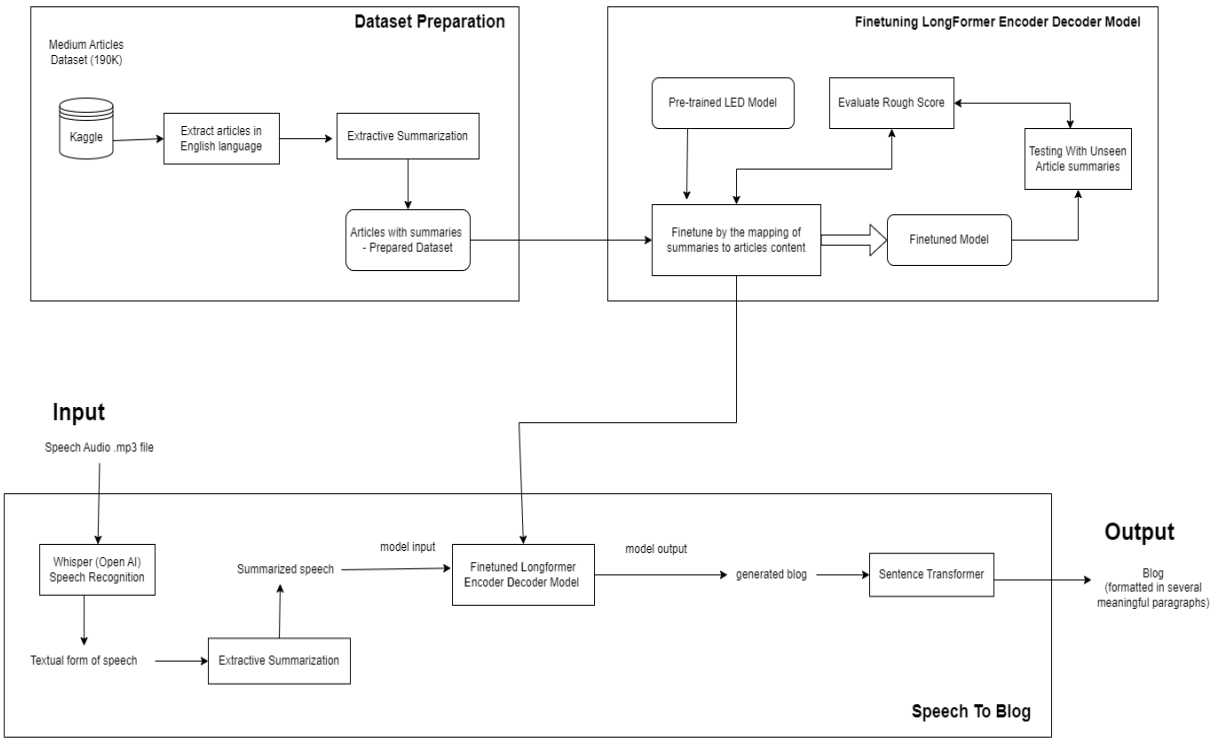
(Speech to Blog project - 100% implementation)

Team:

Name: Gautham Kumar G
Roll no: 2020103521

Name: Senthamizhan B
Roll no: 2020103568

Description:



The project's aim is to convert speech (like lectures) to a well formatted blog. For this project, we've developed the above architecture. Longformer Encoder Decoder

Model has been used and fine tuned for generating the content of the blog. The same model is finetuned for another task of generating headings for paragraphs of the blog. The complete process is described below: (Note that only part of the codes are provided as screenshots, the complete code files are available in the “all code” folder)

Firstly the provided speech will be converted into text with the help of OpenAI’s whisper API for speech recognition. The resultant transcript is well formatted with punctuations. The summary for this transcript is generated via NLTK and Genism libraries in python (extractive summarization). The transcript summary is given as input to the fine tuned model1 to generate new blog content. Result of this model would be brand new content with sentences of the input (summary) included in it. But since it is one single long paragraph, sentence transformers are used to split the content into multiple paragraphs. Finally small paragraphs are grouped together and subheadings are generated with the help of fine tuned model2.

Creating dataset for generating blog:

Medium articles dataset is collected from kaggle. It had around 1,90,000 articles. Extractive summarization is done for each article and summary column is created for the dataset.

```
count = 0
def generateSummary(blog):
    global count
    count += 1
    print("Summarising blog ", count)
    try:
        sentences=sent_tokenize(blog)
        sentences_clean=[re.sub(r'[^\w\s]','',sentence.lower()) for sentence in sentences]
        stop_words = stopwords.words('english')
        sentence_tokens=[[words for words in sentence.split(' ') if words not in stop_words]
for sentence in sentences_clean]
        w2v=Word2Vec(sentence_tokens,vector_size=1,min_count=1,epochs=1000)
        sentence_embeddings=[[w2v.wv.get_vector(word)[0] for word in words] for words in
sentence_tokens]
        max_len=max([len(tokens) for tokens in sentence_tokens])
        sentence_embeddings=[np.pad(embedding,(0,max_len-len(embedding)),'constant') for
embedding in sentence_embeddings]
        similarity_matrix = np.zeros([len(sentence_tokens), len(sentence_tokens)])
        for i,row_embedding in enumerate(sentence_embeddings):
```

```
            for j,column_embedding in enumerate(sentence_embeddings):

similarity_matrix[i][j]=1-spatial.distance.cosine(row_embedding,column_embedding)
        nx_graph = nx.from_numpy_array(similarity_matrix)
        scores = nx.pagerank(nx_graph, max_iter=600)
        top_sentence={sentence:scores[index] for index,sentence in enumerate(sentences)}
        sentNeeded = round(0.25 * len(sentences)) - 1
        top=dict(sorted(top_sentence.items(), key=lambda x: x[1],
reverse=True)[:sentNeeded])
        summary = ""
        for sent in sentences:
            if sent in top.keys():
                summary += sent
        return summary
    except:
        return float("NaN")

import math
filename = "articlesSet.csv"
fields = ['title', 'summary', 'content']
# writing to csv file
with open(filename, 'a') as csvfile:
    # creating a csv writer object
    csvwriter = csv.writer(csvfile)

    # writing the fields
    csvwriter.writerow(fields)

    def callback(row):
        summary = generateSummary(row['text'])
        if(type(summary) != str):
            return
        rows = [row['title'], summary, row['text']]
        csvwriter.writerow(rows)
    df.apply(callback, axis=1)
```

Fine Tuning LED for generating blog:

Now, the LongFormer Encoder Decoder model has to be finetuned by giving the summary of a blog as input and the actual blog content as output.

```
from transformers import AutoModelForSeq2SeqLM
led = AutoModelForSeq2SeqLM.from_pretrained("allenai/led-base-16384",
gradient_checkpointing=True, use_cache=False)
led.config.num_beams = 2
led.config.max_length = 1024
led.config.min_length = 512
```

```
led.config.length_penalty = 2.0
led.config.early_stopping = True
led.config.no_repeat_ngram_size = 3
rouge = load_metric("rouge")
def compute_metrics(pred):
    labels_ids = pred.label_ids
    pred_ids = pred.predictions

    pred_str = tokenizer.batch_decode(pred_ids, skip_special_tokens=True)
    labels_ids[labels_ids == -100] = tokenizer.pad_token_id
    label_str = tokenizer.batch_decode(labels_ids, skip_special_tokens=True)

    rouge_output = rouge.compute(
        predictions=pred_str, references=label_str, rouge_types=["rouge2"]
    )["rouge2"].mid

    return {
        "rouge2_precision": round(rouge_output.precision, 4),
        "rouge2_recall": round(rouge_output.recall, 4),
        "rouge2_fmeasure": round(rouge_output.fmeasure, 4),
    }
```

```
from transformers import Seq2SeqTrainer, Seq2SeqTrainingArguments
training_args = Seq2SeqTrainingArguments(
    predict_with_generate=True,
    evaluation_strategy="steps",
    per_device_train_batch_size=batch_size,
    per_device_eval_batch_size=batch_size,
    output_dir="./",
    logging_steps=5,
    eval_steps=10,
    save_steps=10,
    save_total_limit=2,
    gradient_accumulation_steps=4,
    num_train_epochs=1
)
trainer = Seq2SeqTrainer(
    model=led,
    tokenizer=tokenizer,
    args=training_args,
    compute_metrics=compute_metrics,
    train_dataset=train_dataset,
    eval_dataset=val_dataset,
)
trainer.train(resume_from_checkpoint=True)
```

Speech Recognition:

We've used OpenAI's whisper API for speech recognition. We've obtained API key in openai portal by creating a free trial account.

```
import openai
openai.api_key = "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"
audio_file= open("/content/sampleSpeech.mp3", "rb")
transcript = openai.Audio.transcribe("whisper-1", audio_file)
```

Summary for transcript:

Using libraries like NLTK and Genism, we were able to generate an extractive summary of the transcript. This means the summary contains the exact sentences from speech, but it has only important sentences while conveying the same context as the original content.

```
import numpy as np
import pandas as pd
import nltk
import re
from nltk.tokenize import sent_tokenize
from nltk.corpus import stopwords
from gensim.models import Word2Vec
from scipy import spatial
import networkx as nx
import csv
nltk.download('punkt')
nltk.download('stopwords')
```

```
def generateSummary(transcript):
    print("Summarizing speech...")
    sentences=sent_tokenize(transcript)
    sentences_clean=[re.sub(r'[^\w\s]','',sentence.lower()) for sentence in sentences]
    stop_words = stopwords.words('english')
    sentence_tokens=[[words for words in sentence.split(' ') if words not in stop_words] for
sentence in sentences_clean]
    w2v=Word2Vec(sentence_tokens,vector_size=1,min_count=1,epochs=1000)
    sentence_embeddings=[[w2v.wv.get_vector(word)[0] for word in words] for words in
sentence_tokens]
    max_len=max([len(tokens) for tokens in sentence_tokens])
    sentence_embeddings=[np.pad(embedding,(0,max_len-len(embedding)),'constant') for
```

```
embedding in sentence_embeddings]
    similarity_matrix = np.zeros([len(sentence_tokens), len(sentence_tokens)])
    for i,row_embedding in enumerate(sentence_embeddings):
        for j,column_embedding in enumerate(sentence_embeddings):

similarity_matrix[i][j]=1-spatial.distance.cosine(row_embedding,column_embedding)
    nx_graph = nx.from_numpy_array(similarity_matrix)
    scores = nx.pagerank(nx_graph, max_iter=600)
    top_sentence={sentence:scores[index] for index,sentence in enumerate(sentences)}
    sentNeeded = round(0.25 * len(sentences)) - 1
    top=dict(sorted(top_sentence.items(), key=lambda x: x[1], reverse=True)[:sentNeeded])
    summary = ""
    for sent in sentences:
        if sent in top.keys():
            summary += sent
    return summary
```

Using fine tuned LED for generating blog:

```
from datasets import load_metric
import torch

from datasets import load_dataset, load_metric
from transformers import LEDTokenizer, LEDForConditionalGeneration

# Load tokenizer
tokenizer = LEDTokenizer.from_pretrained("/content/checkpoint-210")
model =
LEDForConditionalGeneration.from_pretrained("/content/checkpoint-210").to("cuda").half()


def generate_answer(batch):
  inputs_dict = tokenizer(batch["summary"], padding="max_length", max_length=1024,
return_tensors="pt", truncation=True)
  input_ids = inputs_dict.input_ids.to("cuda")
  attention_mask = inputs_dict.attention_mask.to("cuda")
  global_attention_mask = torch.zeros_like(attention_mask)
#   # put global attention on <s> token
#   global_attention_mask[:, 0] = 1

  predicted_abstract_ids = model.generate(input_ids, attention_mask=attention_mask,
global_attention_mask=global_attention_mask)
  batch["predicted_content"] = tokenizer.batch_decode(predicted_abstract_ids,
skip_special_tokens=True)
  return batch


result = df_test.map(generate_answer, batched=True, batch_size=2)
generated_blog = result['predicted_content']
```

Splitting blog into paragraphs:

For this task, sentence transformers are used. They will calculate relative minimas, where the context of sentences changes. If the minimas fall below a certain threshold, then there comes the start of a new paragraph.

```
sentences = result['predicted_content'][0].split('. ')
model = SentenceTransformer('all-mpnet-base-v2')
# Determine longest outlier
sentence_length = [len(each) for each in sentences]
long = np.mean(sentence_length) + np.std(sentence_length) *2
# Determine shortest outlier
short = np.mean(sentence_length) - np.std(sentence_length) *2
# Shorten long sentences
text = ''
for each in sentences:
    if len(each) > long:
        # let's replace all the commas with dots
        comma_splitted = each.replace(',', '.')
    else:
        text+= f'{each}. '
sentences = text.split('. ')
# Now let's concatenate short ones
text = ''
for each in sentences:
    if len(each) < short:
        text+= f'{each} '
    else:
        text+= f'{each}. '
```

```
def rev_sigmoid(x:float)->float:
    return (1 / (1 + math.exp(0.5*x)))

def activate_similarities(similarities:np.array, p_size=10)->np.array:
        x = np.linspace(-10,10,p_size)
        y = np.vectorize(rev_sigmoid)

        activation_weights = np.pad(y(x),(0,similarities.shape[0]-p_size))
        diagonals = [similarities.diagonal(each) for each in range(0,similarities.shape[0])]
        diagonals = [np.pad(each, (0,similarities.shape[0]-len(each))) for each in
diagonals]
        diagonals = np.stack(diagonals)
        diagonals = diagonals * activation_weights.reshape(-1,1)
        activated_similarities = np.sum(diagonals, axis=0)
        return activated_similarities

activated_similarities = activate_similarities(similarities, p_size=10)
```

```
fig, ax = plt.subplots()

minmimas = argrelextrema(activated_similarities, np.less, order=2)
reducedMinmimas = argrelextrema(activated_similarities, np.less, order=7)

sns.lineplot(y=activated_similarities, x=range(len(activated_similarities)),
ax=ax).set_title('Relative minimas');

plt.vlines(x=minmimas, ymin=min(activated_similarities), ymax=max(activated_similarities),
colors='purple', ls='--', lw=1, label='vline_multiple - full height')
```

Splitting blogs into paragraphs with the help of minimas. Here we obtained reduced minimas to group small paragraphs into one and consider them for generating subheading in the following section.

```
heading_split_points = [each for each in reducedMinmimas[0]]
textForHeadings = ''
for num,each in enumerate(sentences):
    if num in heading_split_points:
        textForHeadings+=f'\n\n {each}. '
    else:
        textForHeadings+=f'{each}. '

textForHeadings = textForHeadings.split("\n\n")
i = 1
split_points = [each for each in minmimas[0]]
text = f'{headings[0]}'
for num,each in enumerate(sentences):
  if num in split_points:
    text+=f'\n\n {each}. '
  else:
    text+=f'{each}. '
  if num in heading_split_points:
    text += f'{headings[i]}'
    i += 1
```

Dataset for generating headings:

Wikihow dataset has been scrapped for several thousand articles. Headings and their corresponding paragraphs are obtained as a result.

```
import requests
from bs4 import BeautifulSoup
import re
```

```
import csv

for count in range(2000):

    # URL of the Wikihow page to scrape
    url = 'https://www.wikihow.com/Special:Randomizer'

    # Send an HTTP request to the URL and receive the HTML content
    response = requests.get(url)
    html_content = response.content

    # Parse the HTML content using BeautifulSoup
    soup = BeautifulSoup(html_content, 'html.parser')
    article_title = soup.find('title').text.strip()
    print(article_title+" "+str(count))

    # Extract the subheadings and paragraphs using the appropriate HTML tags
    subheadings = []
    paragraphs = []
    steps = soup.find_all('div', {'class': 'step'})
    for step in steps:
        subheading_element = step.find('b')
        subheading_text = subheading_element.text.strip().replace('\n','')
        subheading_text = subheading_text.encode('ascii', errors='ignore').decode()
        subheading_text = re.sub(r'\[\d+\]', '', subheading_text)
        subheadings.append(subheading_text)
        subheading_element.extract()
        for span_tag in step.find_all('span'):
            span_tag.extract()
        paragraph_text = step.text.strip().replace('\n','').replace(' ', ' ')
        paragraph_text = paragraph_text.encode('ascii', errors='ignore').decode()
        paragraph_text = re.sub(r'\[\d+\]', '', paragraph_text)
        paragraphs.append(paragraph_text)



    with open('wikiHow.csv', mode='a', newline='',encoding='utf-8') as csv_file:
        writer = csv.writer(csv_file)
        for i in range(len(subheadings)):
            writer.writerow([article_title,subheadings[i], paragraphs[i]])
```

Fine tuning LED for generating headings:

This is the same as the previous finetune process for generating blogs from summaries. The input will be a paragraph and the output will be the corresponding heading. Here is a sample output of the model:

```
sample_paragraph = "The reason why I loved the top-down culture at Apple is that important decisions are taken faster. Having an expert giving you green light or not keeps the momentum. How many times in a bottom-up culture do we spend weeks and weeks, sometimes even months, trying to get alignment with +10 people, because every single person needs to agree with the point of view? It is exhausting. So again, my experience is that having that one leader to look up to to help guide decisions is time-saving, it helps us focus on the design craft, instead of project managing"

result["generated_heading"]

# output
['Have an expert to guide you.']
```

Generating headings for blog:

```
def generateHeading(paragraph, headingModelPath):
  data = [paragraph]
  df = pd.DataFrame(data, columns=['paragraph'])
  df['paragraph'][0]
  df_test = Dataset.from_pandas(df)

  # load tokenizer
  headingTokenizer = LEDTokenizer.from_pretrained(headingModelPath)
  headingModel =
LEDForConditionalGeneration.from_pretrained(headingModelPath).to("cuda").half()


  def generate_answer(batch):
    inputs_dict = headingTokenizer(batch["paragraph"], padding="max_length", max_length=512,
return_tensors="pt", truncation=True)
    input_ids = inputs_dict.input_ids.to("cuda")
    attention_mask = inputs_dict.attention_mask.to("cuda")
    global_attention_mask = torch.zeros_like(attention_mask)

    predicted_abstract_ids = headingModel.generate(input_ids, attention_mask=attention_mask,
global_attention_mask=global_attention_mask)
    batch["generated_heading"] = headingTokenizer.batch_decode(predicted_abstract_ids,
skip_special_tokens=True)
    return batch


  result = df_test.map(generate_answer, batched=True, batch_size=2)
  return result["generated_heading"][0]
```

```
textForHeadings = textForHeadings.split("\n\n")
headings = []
for i in textForHeadings:
  headings.append(generateHeading(i, heading_model_path))

i = 1
split_points = [each for each in minmimas[0]]
text = f'\033[1m{headings[0][:-1]}:\033[0m \n\n'
for num,each in enumerate(sentences):
  if num in heading_split_points:
    text += f'\n\n \033[1m{headings[i][:-1]}:\033[0m'
    i += 1
  if num in split_points:
    text+=f'\n\n {each}. '
  else:
    text+=f'{each}. '
```

The text will be the final result of our proposed system. A sample is as follows:

Learn about Silicon Valley:

We are a small company that has grown to become one of the largest companies in the world. We have grown to be a global company with over 1.5 million employees, and we have a global presence. We now live on our own campus down the road and in dozens of countries, but the spirit continues. We are a company of over 1 million people, and our mission is to make the world a better place for everyone. We believe that we can do this by creating a world where everyone is connected. Our browser Chrome, a small team's obsession to bring a faster, more secure web to everyone, now has over a billion monthly active users.

More than 1 million students from 11 countries have gone on virtual field trips. Let me tell you a little bit about my personal journey to Silicon Valley from India 22 years ago. I remember in my parents' house in Chennai reading about the invention of the transistor at Bell Labs.

Learn about the history of the Internet:

It was the first transistor in the history of the world, and it was the invention that made the world what it is today. In fact, our most popular products almost all began as big ideas that powers your Twitter feed or your WeChat messages today. But we have also been able to build a world-class network that allows us to connect with people around the world for free. We also have a network of over 100,000 people who are connected to the internet through our network.

This is such an important issue for our company, but also for our country and for a community of entrepreneurs around the globe. We will continue to grow as a company and as a community as we continue to expand our network and expand our presence in the global marketplace. We're also working to bring the Internet to more people, which is why we've created a new generation of mobile devices. We want to make this a reality for all of our users, and that's why we have created a world of virtual reality.

Build a network:

We hope to be the first to do this. We started with a simple idea, a simple concept, that we wanted to build on. We wanted to make it easy for people to connect to the Internet, and so we built a network that would allow us to do just that. We built a web browser that was able to connect people to the web, but we also built a mobile app that could connect people with the internet. We created a virtual world where people could connect with each other, and the internet was a way to connect them. In the same way, we have built apps that revolutionize the way we travel around cities. We can draw a direct line from that invention to the technology that powers the web to people today. We don't need to be an internet company to do that, but rather, we want to be part of the solution.

Tell me about India:

Let you tell you about my experience in India. I was born and raised in a small town in the United States, and I grew up in a family of entrepreneurs. We had a lot of friends and family who helped us build

our company. We were able to create a world that was connected to our world through our technology, and now we are able to do the same with our technology. We live in a world without internet access, but our technology is connected to everything we need to do. We know that the internet is connected, but it is not connected to anything we need. We do not need to have internet access to everything.

We just need to connect and connect. We need to build the internet to everyone. Let the internet be a part of who we are. Let Google connect to us, and Google connect us to the world around us.

Let YouTube connect us with the world through the Internet. Let Facebook connect us through the web. Let Twitter connect us from the internet, and you can connect with us through our social media accounts. Let Instagram connect us on the social media platforms. Let Snapchat connect us.Let Google connect you to your friends. Let Gmail connect you with your friends and your family. Let Yahoo connect you. Let Netflix connect you and your friends with your favorite apps. Let Go of the World! Let Google Connect to the World. Let Me Tell You a Little bit About Silicon Valley From India 22 Years ago.. .