Wiki-How Content Generation: Heading, Summary, and Tags

This document explains a Python script that uses finetuned models to generate headings, summaries, and tags for given paragraphs. The script utilizes two main models: a fine-tuned LED (Longformer Encoder-Decoder) model for heading and summary generation, and a fine-tuned BERT model for tag generation.

Setup and Dependencies

installing necessary libraries:

- transformers: For using pre-trained models
- datasets: For dataset handling
- pandas: For data manipulation
- rouge score: For evaluation (though not used in this snippet)
- torch: PyTorch for deep learning operations

LED Model for Heading and Summary Generation

Model Loading

The script loads a fine-tuned LED model and its tokenizer from a specified checkpoint in Google Drive.

Text Generation Function

The `generate_text` function is defined to generate headings and summaries:

1. It tokenizes the input paragraph.
2. Uses the LED model to generate heading and summary separately.
3. Decodes the generated tokens back into text.

BERT Model for Tag Generation

Model Training

The script includes steps to fine-tune a BERT model for masked language modeling:

1. Loads a pre-trained Polish BERT model.

2. Prepares a dataset from a text file.

3. Sets up training arguments and trains the model.

Tag Generation Function

The `generate_tags` function uses the fine-tuned BERT model:

1. Tokenizes the input paragraph.

2. Masks a token for prediction.

3. Uses the model to predict the masked token.

4. Returns the top 5 predictions as potential tags.

Inference

The script demonstrates how to use these functions:

1. It defines a sample paragraph.

2. Creates a dataset from this paragraph.

3. Generates heading and summary using the LED model.

4. Generates tags using the BERT model.

5. Prints the results.

Key Points

- The LED model is used for generating longer text (headings and summaries).

- The BERT model is used for generating shorter, keyword-like text (tags).

- Both models are fine-tuned on specific tasks before being used for generation.

- The script showcases how different AI models can be combined for a complex text generation task.

Sample images:

Get a random blog post from WikiHow

```
[3]  url = "https://www.wikihow.com/Special:Randomizer"
     response = requests.get(url)
     soup = bs4.BeautifulSoup(response.content, 'html.parser')
```

Web scraping:

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')
    if subheading_element is not None:
        subheading_text = subheading_element.text.strip().replace('\n', '')
        subheading_text = subheading_text.encode('ascii', errors='ignore').decode()
        subheading_text = re.sub(r'\r', '', subheading_text)
        print(subheading_text)
        subheadings.append(subheading_text)

        #  removes titles and extra links
        subheading_element.extract()
        for span_tag in step.find_all('span'):
            span_tag.extract()
```

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')
    if subheading_element is not None:
        subheading_text = subheading_element.text.strip().replace('\n', '')
        subheading_text = subheading_text.encode('ascii', errors='ignore').decode()
        subheading_text = re.sub(r'\r', '', subheading_text)
        print(subheading_text)
        subheadings.append(subheading_text)

        #  removes titles and extra links
        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'\r', '', paragraph_text)
    print(paragraph_text)
    paragraphs.append(paragraph_text)
```

Reading csv

import lib

```
[4]  import pandas as pd
     import matplotlib.pyplot as plt
     from transformers import LEDTokenizer, LEDForConditionalGeneration
```

load csv

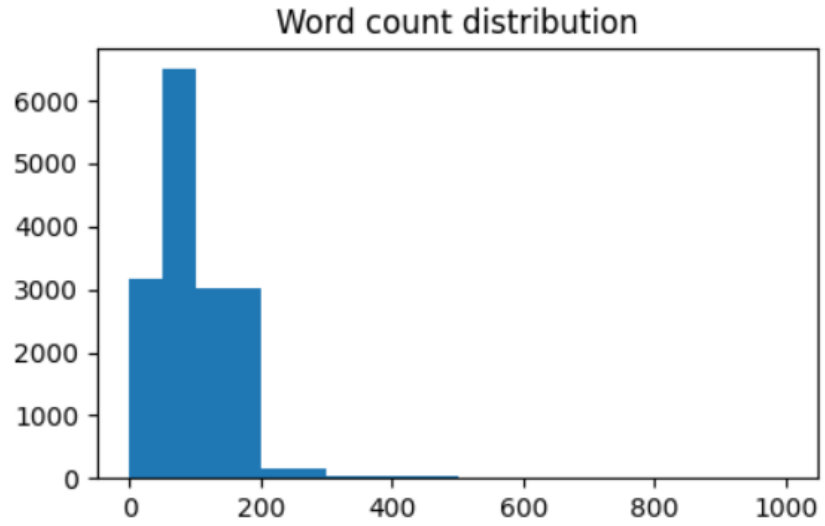
```
file_path = '/content/drive/MyDrive/wikiHow.csv'
df = pd.read_csv(file_path)
```

```python
# Create the plot with a specified figure size
fig = plt.figure(figsize=(5, 3))

# Plot a histogram of the word count distribution, specifying the bin ranges
plt.hist(numOfWords.to_numpy(), bins=[0, 50, 100, 200, 300, 500, 1000])

# Add a title to the plot
plt.title("Word count distribution")

# Show the plot
plt.show()
```



Data Spliting

```python
import numpy as np
train, validate, test = np.split(tempdf.sample(frac=1, random_state=42), [int(.6*len(df)), int(.7*len(df))])
print(train.shape)
print(validate.shape)
print(test.shape)
validate = validate[:20]
print(validate.shape)
```

```
(7722, 4)
(1287, 4)
(3677, 4)
(20, 4)
/usr/local/lib/python3.10/dist-packages/numpy/core/fromnumeric.py:59: FutureWarning: 'DataFrame.swapaxes' is deprecated and will be removed in a
  return bound(*args, **kwds)
```

```
***** Running training *****
  Num examples = 7,722
  Num Epochs = 10
  Instantaneous batch size per device = 16
  Total train batch size (w. parallel, distributed & accumulation) = 64
  Gradient Accumulation steps = 4
  Total optimization steps = 1,200
  Number of trainable parameters = 161,844,480
/usr/local/lib/python3.10/dist-packages/torch/utils/checkpoint.py:295: FutureWarnir
  with torch.enable_grad(), device_autocast_ctx, torch.cpu.amp.autocast(**ctx.cpu_a
```

[81/1200 43:57 < 10:22:45, 0.03 it/s, Epoch 0.66/10]

Step	Training Loss	Validation Loss	Rouge2 Precision	Rouge2 Recall	Rouge2 Fmeasure
10	2.152900	1.785071	0.191700	0.181700	0.182400
20	2.129500	1.727933	0.191200	0.215300	0.199500
30	2.170000	1.719635	0.247100	0.263100	0.250000
40	2.153800	1.725903	0.212100	0.253000	0.217400
50	2.152200	1.740086	0.221900	0.249500	0.220900
60	2.043200	1.752729	0.286600	0.269000	0.270400
70	2.043300	1.717124	0.225600	0.226500	0.218900
80	2.151100	1.700823	0.252700	0.235800	0.237200

```
sample_paragraph = """In recent years, artificial intelligence (AI) has made significa
industries, revolutionizing the way we live and work. From autonomous vehicles to
personalized recommendations, AI-powered solutions have enhanced efficiency
and productivity. However, with these advancements come concerns about ethics,
privacy, and the future of employment. As AI continues to evolve, it is essential to
```

Map: 100% 1/1 [00:00<00:00, 2.65 examples/s]

```
Generated Heading: ['Advancements and ethics in artificial intelligence']
Generated Summary: ['The rapid advancements in artificial intelligence (AI) and its imp
Generated Tags: ['AI', 'automation', 'privacy']
```