DepartmentofComputerScience&Engineering

(DataScience)

Laboratory Manual

Subject: - ACL (PECS7042L) **Semester: - VII**

Class: - T. Y. B. Tech **Experiment No. : - 7**

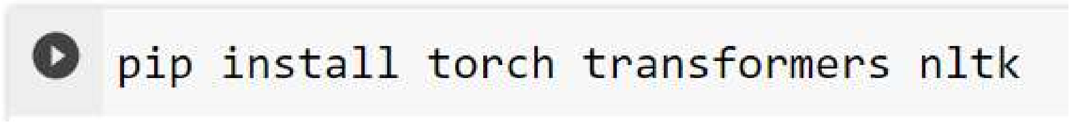
Aim: - Implement text Summarization using BERT.
Requirement: -Python versions 3.7, 3.8, 3.9, 3.10 or 3.11, jupyter notebook

Theory: -

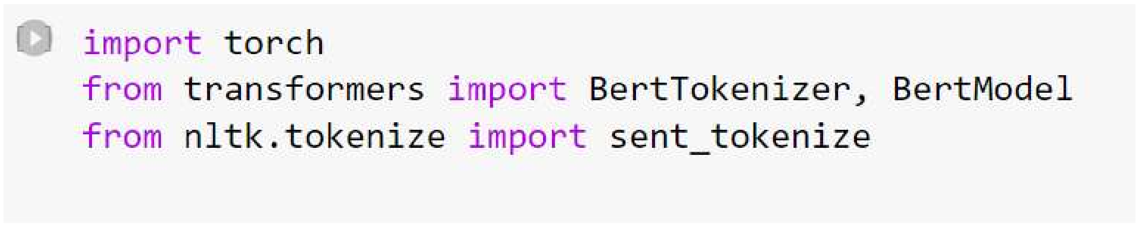
Text summarization is the process of reducing the length of a text document while retaining its important information. This can be achieved through various methods, including extraction-based summarization and abstraction-based summarization. One popular approach for extractive summarization is BERT (Bidirectional Encoder Representations from Transformers), which is a pre-trained language model that can be fine-tuned for text summarization tasks. In this tutorial, we will explore how to use BERT for text summarization in Python.

Prerequisites Before we dive into the implementation, it is essential to have a basic understanding of natural language processing (NLP), Python programming language, and PyTorch library. In addition, it would be helpful to have some experience with BERT and the Hugging Face Transformers library. You can install the required libraries by running the following command in your terminal:

Step 1:

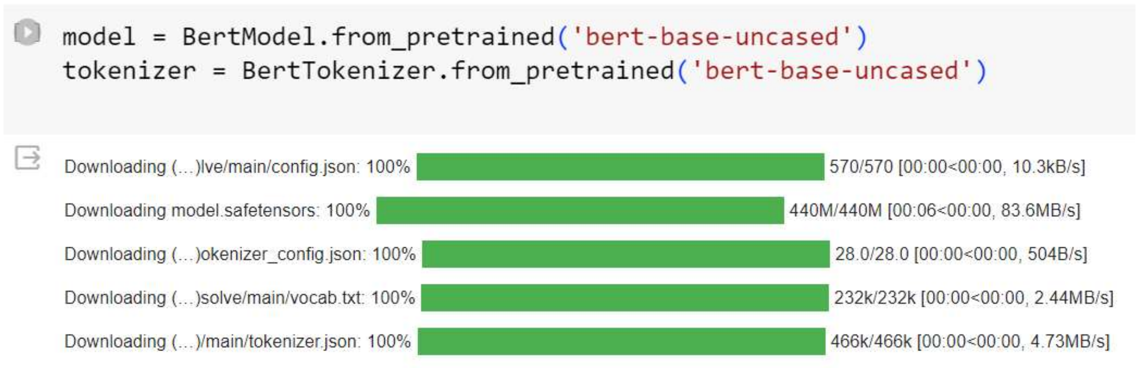


```
pip install torch transformers nltk
```



```
import torch
from transformers import BertTokenizer, BertModel
from nltk.tokenize import sent_tokenize
```

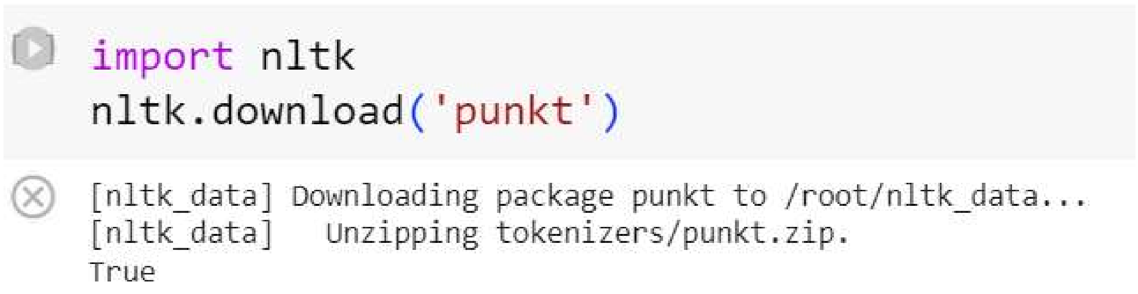
Step 2: Loading the pre-trained BERT model and tokenizer next, we will load the pre-trained BERT model and tokenizer using the Hugging Face transformers library.



```
model = BertModel.from_pretrained('bert-base-uncased')
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
```

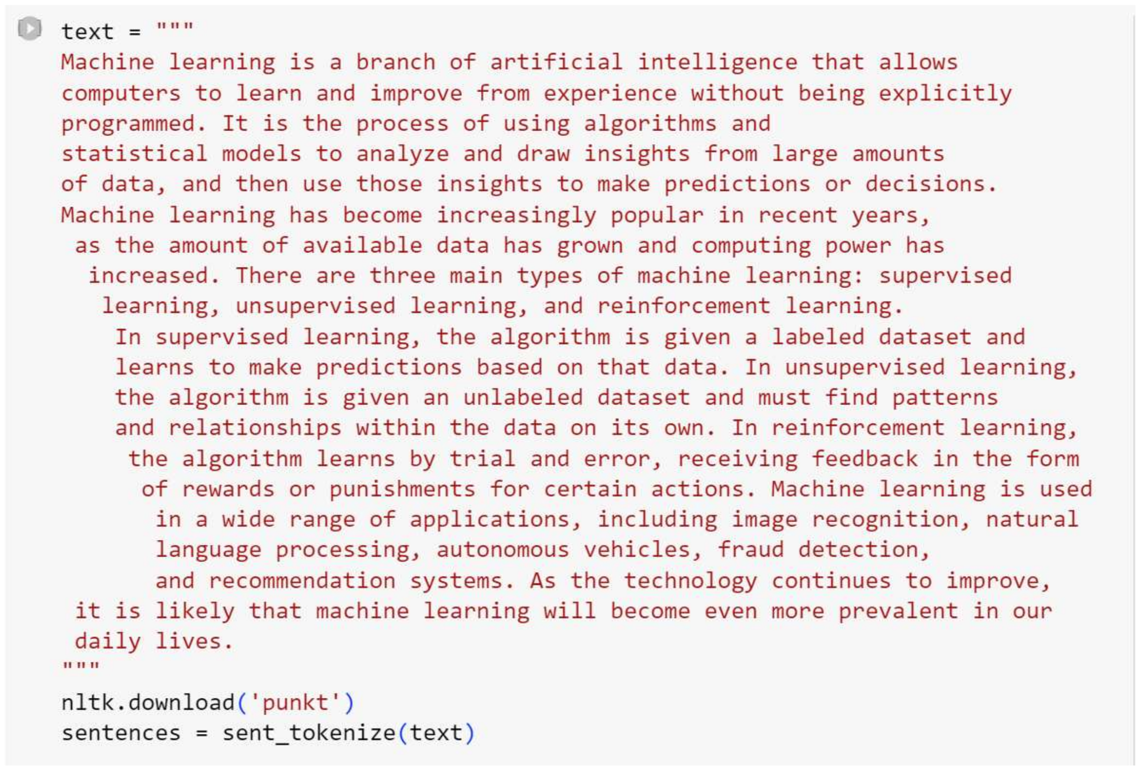
```
Downloading (…)lve/main/config.json: 100%        570/570 [00:00<00:00, 10.3kB/s]
Downloading model.safetensors: 100%              440M/440M [00:06<00:00, 83.6MB/s]
Downloading (…)okenizer_config.json: 100%        28.0/28.0 [00:00<00:00, 504B/s]
Downloading (…)solve/main/vocab.txt: 100%        232k/232k [00:00<00:00, 2.44MB/s]
Downloading (…)/main/tokenizer.json: 100%        466k/466k [00:00<00:00, 4.73MB/s]
```

Step 3: Preprocessing the text Before we can feed the text into the BERT model, we need to preprocess it. First, we will split the text into sentences using the NLTK sent_tokenize function.



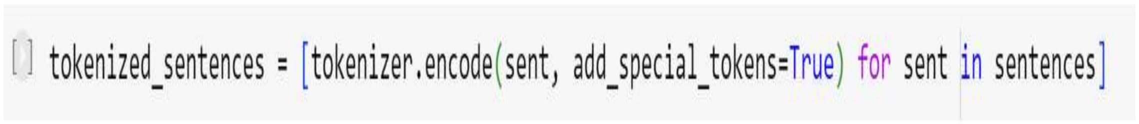
```
import nltk
nltk.download('punkt')
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
True
```



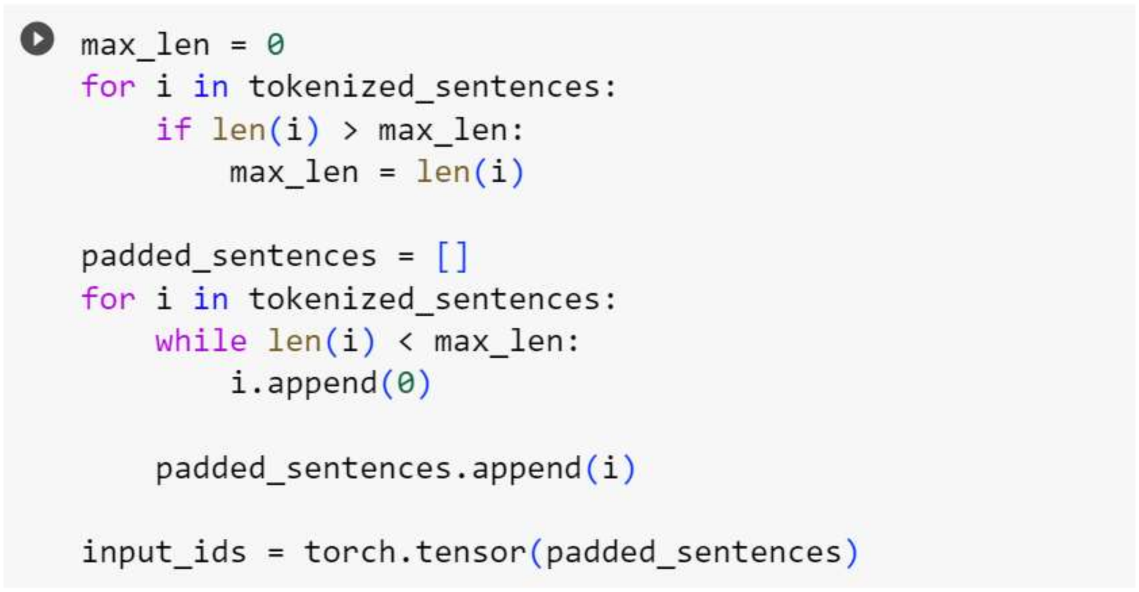
```
text = """
Machine learning is a branch of artificial intelligence that allows
computers to learn and improve from experience without being explicitly
programmed. It is the process of using algorithms and
statistical models to analyze and draw insights from large amounts
of data, and then use those insights to make predictions or decisions.
Machine learning has become increasingly popular in recent years,
 as the amount of available data has grown and computing power has
  increased. There are three main types of machine learning: supervised
   learning, unsupervised learning, and reinforcement learning.
    In supervised learning, the algorithm is given a labeled dataset and
    learns to make predictions based on that data. In unsupervised learning,
    the algorithm is given an unlabeled dataset and must find patterns
    and relationships within the data on its own. In reinforcement learning,
     the algorithm learns by trial and error, receiving feedback in the form
      of rewards or punishments for certain actions. Machine learning is used
       in a wide range of applications, including image recognition, natural
       language processing, autonomous vehicles, fraud detection,
       and recommendation systems. As the technology continues to improve,
 it is likely that machine learning will become even more prevalent in our
 daily lives.
"""
nltk.download('punkt')
sentences = sent_tokenize(text)
```

Next, we will tokenize each sentence using the BERT tokenizer.

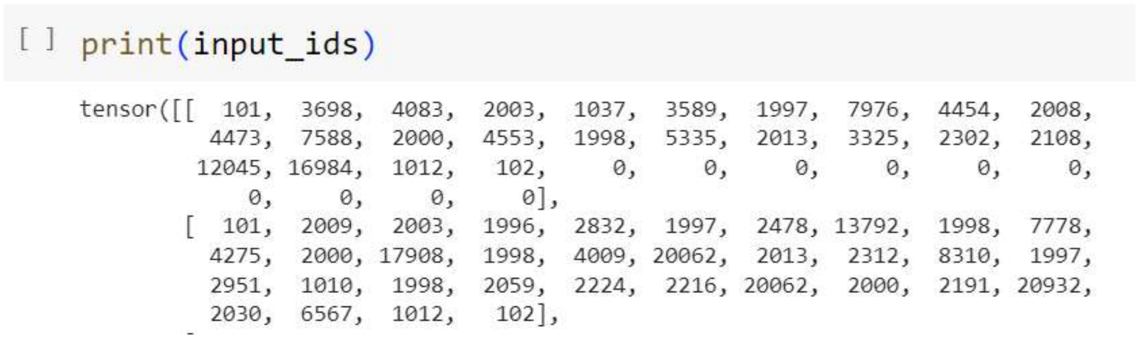


```
[ ] tokenized_sentences = [tokenizer.encode(sent, add_special_tokens=True) for sent in sentences]
```

Step 4: Encoding the input To feed the tokenized sentences into the BERT model, we need to encode them using the BERT tokenizer. We also need to add special tokens such as [CLS] (for the start of the sentence) and [SEP] (for the end of the sentence) to each input.



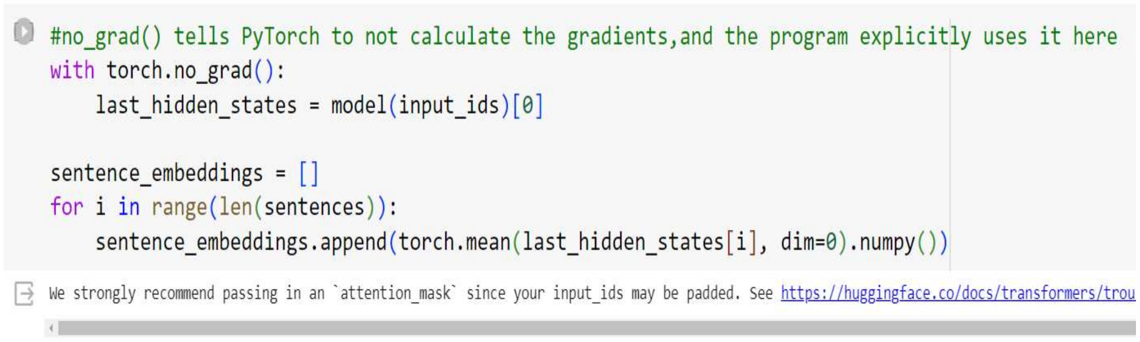
```
max_len = 0
for i in tokenized_sentences:
    if len(i) > max_len:
        max_len = len(i)

padded_sentences = []
for i in tokenized_sentences:
    while len(i) < max_len:
        i.append(0)

    padded_sentences.append(i)

input_ids = torch.tensor(padded_sentences)
```



```
[ ] print(input_ids)
```

```
tensor([[  101,  3698,  4083,  2003,  1037,  3589,  1997,  7976,  4454,  2008,
          4473,  7588,  2000,  4553,  1998,  5335,  2013,  3325,  2302,  2108,
         12045, 16984,  1012,   102,     0,     0,     0,     0,     0,     0,
             0,     0,     0,     0],
        [  101,  2009,  2003,  1996,  2832,  1997,  2478, 13792,  1998,  7778,
          4275,  2000, 17908,  1998,  4009, 20062,  2013,  2312,  8310,  1997,
          2951,  1010,  1998,  2059,  2224,  2216, 20062,  2000,  2191, 20932,
          2030,  6567,  1012,   102],
```

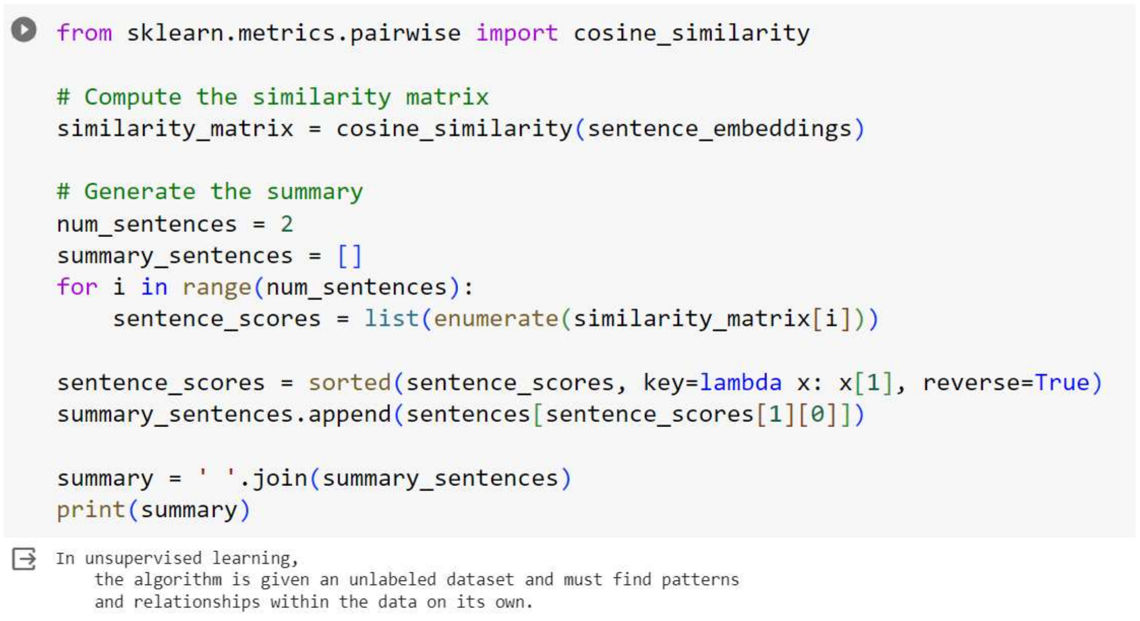
Step 5: Generating the sentence embeddings. Once we have encoded the input, we can feed it into the BERT model to generate sentence embeddings. We will use the last hidden state of the BERT model as the sentence embedding.



```
#no_grad() tells PyTorch to not calculate the gradients,and the program explicitly uses it here
with torch.no_grad():
    last_hidden_states = model(input_ids)[0]

sentence_embeddings = []
for i in range(len(sentences)):
    sentence_embeddings.append(torch.mean(last_hidden_states[i], dim=0).numpy())
```

```
We strongly recommend passing in an `attention_mask` since your input_ids may be padded. See https://huggingface.co/docs/transformers/trou
```

Step 6: Summarizing the text finally, we can use sentence embeddings to summarize the text. One way to do this is to compute the similarity between each sentence and the other sentences and select the sentences with the highest similarity scores. We will use the cosine similarity measure for this.



```python
from sklearn.metrics.pairwise import cosine_similarity

# Compute the similarity matrix
similarity_matrix = cosine_similarity(sentence_embeddings)

# Generate the summary
num_sentences = 2
summary_sentences = []
for i in range(num_sentences):
    sentence_scores = list(enumerate(similarity_matrix[i]))

sentence_scores = sorted(sentence_scores, key=lambda x: x[1], reverse=True)
summary_sentences.append(sentences[sentence_scores[1][0]])

summary = ' '.join(summary_sentences)
print(summary)
```

```
In unsupervised learning,
    the algorithm is given an unlabeled dataset and must find patterns
    and relationships within the data on its own.
```