Problem Overview

The project focuses on document similarity using language models to identify the most relevant document or sentence based on a given input text. This is particularly useful in plagiarism detection, information retrieval, and semantic search applications. Our team focused on exploring various methods for identifying semantic similarity between sentences. We experimented with multiple approaches to enhance the accuracy and effectiveness of similarity detection:

Chosen Solution: Using BERT for Document Similarity

To address the document similarity task, we propose using BERT (Bidirectional Encoder Representations from Transformers). BERT's contextual embeddings capture the semantic meaning of text by understanding the relationships between words in a sentence. We leverage pre-trained BERT models and fine-tune them for our specific task of document similarity.

Why BERT?

1. Contextual Understanding: Unlike traditional word embeddings like Word2Vec, BERT captures the contextual meaning of words based on the entire sentence.
2. Bidirectional Encoding: BERT reads text in both directions (left-to-right and right-to-left), improving the quality of embeddings.
3. Flexibility: Pre-trained on massive datasets, BERT can be fine-tuned for specific tasks with minimal labeled data.

BERT Embeddings for Semantic Similarity

- We started by generating embeddings using the basic BERT model and calculating similarity scores between sentences based on these embeddings

Text Summarization and Similarity Analysis

- Next, we incorporated a combination of BART (for text summarization) and BERT (for embedding generation) to explore sentence similarity between summarized texts.

BERT Classification Model

- Finally, we trained a BERT-based classification model to predict similarity labels ("similar" or "not similar") for sentence pairs.

Datasets Used

Synthetic Dataset

- We generated a custom dataset stored in the Synthetic folder. This dataset includes original and paraphrased sentence pairs derived from Pride and Prejudice by Jane Austen.

STS Benchmark Dataset

- The STS Benchmark dataset was used to train our BERT classification model for predicting sentence similarity.

In [ ]:
```
import numpy as np  # linear algebra
import pandas as pd  # data processing, CSV file I/O (e.g. pd.read_csv)
import os

for dirname, _, filenames in os.walk("/kaggle/input"):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

In [ ]:
```
!pip install -U sentence-transformers
```

```
Collecting sentence-transformers
  Downloading sentence_transformers-3.3.1-py3-none-any.whl.metadata (10 k
B)
Requirement already satisfied: transformers<5.0.0,>=4.41.0 in /opt/conda/l
ib/python3.10/site-packages (from sentence-transformers) (4.45.1)
Requirement already satisfied: tqdm in /opt/conda/lib/python3.10/site-pack
ages (from sentence-transformers) (4.66.4)
Requirement already satisfied: torch>=1.11.0 in /opt/conda/lib/python3.10/
site-packages (from sentence-transformers) (2.4.0)
Requirement already satisfied: scikit-learn in /opt/conda/lib/python3.10/s
ite-packages (from sentence-transformers) (1.2.2)
Requirement already satisfied: scipy in /opt/conda/lib/python3.10/site-pac
kages (from sentence-transformers) (1.14.1)
Requirement already satisfied: huggingface-hub>=0.20.0 in /opt/conda/lib/p
ython3.10/site-packages (from sentence-transformers) (0.25.1)
Requirement already satisfied: Pillow in /opt/conda/lib/python3.10/site-pa
ckages (from sentence-transformers) (10.3.0)
Requirement already satisfied: filelock in /opt/conda/lib/python3.10/site-
packages (from huggingface-hub>=0.20.0->sentence-transformers) (3.15.1)
Requirement already satisfied: fsspec>=2023.5.0 in /opt/conda/lib/python3.
10/site-packages (from huggingface-hub>=0.20.0->sentence-transformers) (20
24.6.1)
Requirement already satisfied: packaging>=20.9 in /opt/conda/lib/python3.1
```

```
0/site-packages (from huggingface-hub>=0.20.0->sentence-transformers) (21.
3)
Requirement already satisfied: pyyaml>=5.1 in /opt/conda/lib/python3.10/si
te-packages (from huggingface-hub>=0.20.0->sentence-transformers) (6.0.2)
Requirement already satisfied: requests in /opt/conda/lib/python3.10/site-
packages (from huggingface-hub>=0.20.0->sentence-transformers) (2.32.3)
Requirement already satisfied: typing-extensions>=3.7.4.3 in /opt/conda/li
b/python3.10/site-packages (from huggingface-hub>=0.20.0->sentence-transfo
rmers) (4.12.2)
Requirement already satisfied: sympy in /opt/conda/lib/python3.10/site-pac
kages (from torch>=1.11.0->sentence-transformers) (1.13.3)
Requirement already satisfied: networkx in /opt/conda/lib/python3.10/site-
packages (from torch>=1.11.0->sentence-transformers) (3.3)
Requirement already satisfied: jinja2 in /opt/conda/lib/python3.10/site-pa
ckages (from torch>=1.11.0->sentence-transformers) (3.1.4)
Requirement already satisfied: numpy>=1.17 in /opt/conda/lib/python3.10/si
te-packages (from transformers<5.0.0,>=4.41.0->sentence-transformers) (1.2
6.4)
Requirement already satisfied: regex!=2019.12.17 in /opt/conda/lib/python
3.10/site-packages (from transformers<5.0.0,>=4.41.0->sentence-transformer
s) (2024.5.15)
Requirement already satisfied: safetensors>=0.4.1 in /opt/conda/lib/python
3.10/site-packages (from transformers<5.0.0,>=4.41.0->sentence-transformer
s) (0.4.5)
Requirement already satisfied: tokenizers<0.21,>=0.20 in /opt/conda/lib/py
thon3.10/site-packages (from transformers<5.0.0,>=4.41.0->sentence-transfo
rmers) (0.20.0)
Requirement already satisfied: joblib>=1.1.1 in /opt/conda/lib/python3.10/
site-packages (from scikit-learn->sentence-transformers) (1.4.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in /opt/conda/lib/pyth
on3.10/site-packages (from scikit-learn->sentence-transformers) (3.5.0)
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in /opt/conda/lib/
python3.10/site-packages (from packaging>=20.9->huggingface-hub>=0.20.0->s
entence-transformers) (3.1.2)
Requirement already satisfied: MarkupSafe>=2.0 in /opt/conda/lib/python3.1
0/site-packages (from jinja2->torch>=1.11.0->sentence-transformers) (2.1.
5)
Requirement already satisfied: charset-normalizer<4,>=2 in /opt/conda/lib/
python3.10/site-packages (from requests->huggingface-hub>=0.20.0->sentence
-transformers) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /opt/conda/lib/python3.10/s
ite-packages (from requests->huggingface-hub>=0.20.0->sentence-transformer
s) (3.7)
Requirement already satisfied: urllib3<3,>=1.21.1 in /opt/conda/lib/python
3.10/site-packages (from requests->huggingface-hub>=0.20.0->sentence-trans
formers) (1.26.18)
Requirement already satisfied: certifi>=2017.4.17 in /opt/conda/lib/python
3.10/site-packages (from requests->huggingface-hub>=0.20.0->sentence-trans
formers) (2024.8.30)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /opt/conda/lib/python
3.10/site-packages (from sympy->torch>=1.11.0->sentence-transformers) (1.
3.0)
Downloading sentence_transformers-3.3.1-py3-none-any.whl (268 kB)
```

```
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 268.8/268.8 kB 7.2 MB/s eta 0:
00:00:00:01
Installing collected packages: sentence-transformers
Successfully installed sentence-transformers-3.3.1
```

In [ ]:
```
# loading a synthetic dataset into a pandas DataFrame named df.
df = pd.read_csv(
    "https://raw.githubusercontent.com/Afag-Ramazanova/Document_Similarit
)
```

In [ ]:
```
res_l = df.loc[:, ["doc1", "doc2"]].values.T
```

Basic Model using BERT for Similarity

Importing Hugging Face's `transformers` and `torch` libraries to work with pre-trained models and sklearn for cosine similarity calculations.

In [ ]:
```
from transformers import AutoTokenizer, AutoModel
import torch
from sklearn.metrics.pairwise import cosine_similarity
```

Here we implemented straightforward methods to calculate semantic similarity between pairs of sentences. These techniques rely on pre-trained embeddings and statistical measures rather than training a machine learning model. Also we are converting the token embeddings into a single sentence embedding using mean pooling. The function *outputs* the similarity score, providing an unsupervised assessment of how similar the two sentences are.

Using this function:

- We set a baseline for performance before advancing to more complex, trainable models.

In [ ]:
```
def compute_similarity(sentences, model_name="bert-base-uncased", max_len
    """
    Compute the cosine similarity between the first sentence and all othe
    Args:
        sentences (list of list of str): A list of sentences where the fi
        model_name (str): The pre-trained BERT model name (default: 'bert
        max_length (int): Maximum sequence length for tokenization (defau
    Returns:
        numpy.ndarray: Cosine similarity values between the first sentenc
    """

    # Load tokenizer and model
    tokenizer = AutoTokenizer.from_pretrained(model_name)
    model = AutoModel.from_pretrained(model_name)

    # Tokenization and input preparation
```

```
    tokens = {"input_ids": [], "attention_mask": []}

    for sentence in sentences:
        new_tokens = tokenizer.encode_plus(
            "\n".join(sentence),
            max_length=max_length,
            truncation=True,
            padding="max_length",
            return_tensors="pt",
        )

        tokens["input_ids"].append(new_tokens["input_ids"][0])
        tokens["attention_mask"].append(new_tokens["attention_mask"][0])

    tokens["input_ids"] = torch.stack(tokens["input_ids"])
    tokens["attention_mask"] = torch.stack(tokens["attention_mask"])

    # Generate embeddings
    outputs = model(
        **tokens
    )  # Passing the input tensors through the BERT model to generate emb
    embeddings = (
        outputs.last_hidden_state
    )  # A tensor that contains the embeddings of all tokens for all sent
    attention = tokens[
        "attention_mask"
    ]  # Used later to ignore padding tokens during the computation.

    # Mask embeddings
    mask = (
        attention.unsqueeze(-1).expand(embeddings.shape).float()
    )  # extra dimension to the attention mask to match the shape of embe
    # repeats the mask across the embedding dimensions.
    masked_embeddings = embeddings * mask  # nullify padding embeddings

    # Compute mean pooling
    summed = torch.sum(
        masked_embeddings, dim=1
    )  # sum of token embeddings for each sentence along the token dimens
    counts = torch.clamp(
        mask.sum(dim=1), min=1e-9
    )  # counts the number of valid (non-padding) tokens & prevent divisi
    mean_pooled = summed / counts
    mean_pooled = (
        mean_pooled.detach().numpy()
    )  # divides the sum by # of valid tokens to compute the average embe

    # cosine similarity
    similarity = cosine_similarity(
        [mean_pooled[0]], mean_pooled[1:]
    )  # calculating the similarity
    return similarity
```

In [ ]:
```
res_similarity = compute_similarity(res_l)[0]
print("Similarity score:" + str(res_similarity))
```

```
tokenizer_config.json:   0%|          | 0.00/48.0 [00:00<?, ?B/s]
config.json:   0%|          | 0.00/570 [00:00<?, ?B/s]
vocab.txt:   0%|          | 0.00/232k [00:00<?, ?B/s]
tokenizer.json:   0%|          | 0.00/466k [00:00<?, ?B/s]
/opt/conda/lib/python3.10/site-packages/transformers/tokenization_utils_base.py:1617: FutureWarning: `clean_up_tokenization_spaces` was not set. It will be set to `True` by default. This behavior will be deprecated in transformers v4.45, and will be then set to `False` by default. For more details check this issue: https://github.com/huggingface/transformers/issues/31884
  warnings.warn(
model.safetensors:   0%|          | 0.00/440M [00:00<?, ?B/s]
Similarity score:[0.9609797]
```

Document Summarization approach with BART & BERT & Cosine Similarity

In this section, the focus is on using document summarization as a preprocessing step to enhance the semantic similarity analysis. Using BART (Bidirectional and Auto-Regressive Transformers), a model specialized for sequence-to-sequence tasks like text summarization, to create concise versions of the input documents.

Process:

- The input sentences or paragraphs are fed into a pre-trained BART model to generate summarized versions.

- This reduces the content to its essential meaning, removing redundancies and irrelevant details.

- Each summarized document is passed through BERT to extract dense numerical embeddings representing the document's semantic content.

- The embeddings are compared using cosine similarity, which quantifies the closeness of the two vectors in the embedding space.

In [ ]:
```
from transformers import AutoTokenizer, AutoModelForSeq2SeqLM
from sentence_transformers import SentenceTransformer
from sklearn.metrics.pairwise import cosine_similarity

# Load pre-trained summarization model
model_name = "facebook/bart-large-cnn"
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForSeq2SeqLM.from_pretrained(model_name)

# Input text
summary_res = []

for i in range(2):
```

```
    text = ";".join(res_l[i])

    # Tokenize and summarize
    inputs = tokenizer.encode(
        "summarize: " + text, return_tensors="pt", max_length=1024, trunc
    )

    summary_ids = model.generate(
        inputs,
        max_length=512,
        min_length=40,
        length_penalty=2.0,
        num_beams=4,
        early_stopping=True,
    )

    summary = tokenizer.decode(summary_ids[0], skip_special_tokens=True)
    summary_res.append(summary)


embedder_model_name = "sentence-transformers/bert-base-nli-mean-tokens"

embedder_model = SentenceTransformer(embedder_model_name)
embeding_summary = embedder_model.encode(summary_res)

similarity_summary = cosine_similarity([embeding_summary[0]], embeding_su

similarity_summary = cosine_similarity([embeding_summary[0]], embeding_su


print("Similarity Percentage = ", similarity_summary[0][0] * 100)
```

```
config.json:   0%|          | 0.00/1.58k [00:00<?, ?B/s]
vocab.json:   0%|          | 0.00/899k [00:00<?, ?B/s]
merges.txt:   0%|          | 0.00/456k [00:00<?, ?B/s]
tokenizer.json:   0%|          | 0.00/1.36M [00:00<?, ?B/s]
/opt/conda/lib/python3.10/site-packages/transformers/tokenization_utils_ba
se.py:1617: FutureWarning: `clean_up_tokenization_spaces` was not set. It
will be set to `True` by default. This behavior will be deprecated in tran
sformers v4.45, and will be then set to `False` by default. For more detai
ls check this issue: https://github.com/huggingface/transformers/issues/31
884
  warnings.warn(
model.safetensors:   0%|          | 0.00/1.63G [00:00<?, ?B/s]
generation_config.json:   0%|          | 0.00/363 [00:00<?, ?B/s]
modules.json:   0%|          | 0.00/229 [00:00<?, ?B/s]
config_sentence_transformers.json:   0%|          | 0.00/122 [00:00<?, ?B/
s]
README.md:   0%|          | 0.00/3.99k [00:00<?, ?B/s]
sentence_bert_config.json:   0%|          | 0.00/53.0 [00:00<?, ?B/s]
config.json:   0%|          | 0.00/625 [00:00<?, ?B/s]
model.safetensors:   0%|          | 0.00/438M [00:00<?, ?B/s]
tokenizer_config.json:   0%|          | 0.00/399 [00:00<?, ?B/s]
vocab.txt:   0%|          | 0.00/232k [00:00<?, ?B/s]
```

```
tokenizer.json:   0%|          | 0.00/466k [00:00<?, ?B/s]
added_tokens.json:   0%|          | 0.00/2.00 [00:00<?, ?B/s]
special_tokens_map.json:   0%|          | 0.00/112 [00:00<?, ?B/s]
1_Pooling/config.json:   0%|          | 0.00/190 [00:00<?, ?B/s]
Batches:   0%|          | 0/1 [00:00<?, ?it/s]
Similarity Percentage =  67.38548874855042
```

Bert Semantic Classification Model

In [ ]:
```
import numpy as np
import pandas as pd
import tensorflow as tf
import transformers
import pandas as pd
```

Pre-Processing Data

This code loads training, validation, and testing datasets for the Semantic Textual Similarity Benchmark (STS):

- `train_df` : Training data.
- `valid_df` : Validation data.
- `test_df` : Test data.

In [ ]:
```
train_df = pd.read_csv(
    "https://raw.githubusercontent.com/Afag-Ramazanova/Document_Similarit
)

valid_df = pd.read_csv(
    "https://raw.githubusercontent.com/Afag-Ramazanova/Document_Similarit
)

test_df = pd.read_csv(
    "https://raw.githubusercontent.com/Afag-Ramazanova/Document_Similarit
)

# Shape of the data
print(f"Total train samples : {train_df.shape[0]}")
print(f"Total validation samples: {valid_df.shape[0]}")
print(f"Total test samples: {valid_df.shape[0]}")
```

```
Total train samples : 5749
Total validation samples: 1500
Total test samples: 1500
```

Here we are labeling each sentence pair as "similar" or "not similar" based on the score (its values range from 0 to 5.0) column. A score of 3 or more is considered "**similar**".

In [ ]:
```
train_df["score_classification"] = train_df["score"].apply(
    lambda x: "similar" if x >= 3 else "not similar"
)

valid_df["score_classification"] = valid_df["score"].apply(
    lambda x: "similar" if x >= 3 else "not similar"
)

test_df["score_classification"] = test_df["score"].apply(
    lambda x: "similar" if x >= 3 else "not similar"
)
```

Converting the similarity labels into binary values (0: not similar, 1: similar).

In [ ]:
```
train_df["label"] = train_df["score_classification"].apply(
    lambda x: 0 if x == "not similar" else 1
)

y_train = tf.keras.utils.to_categorical(train_df.label, num_classes=2)

valid_df["label"] = valid_df["score_classification"].apply(
    lambda x: 0 if x == "not similar" else 1
)

y_val = tf.keras.utils.to_categorical(valid_df.label, num_classes=2)

test_df["label"] = test_df["score_classification"].apply(
    lambda x: 0 if x == "not similar" else 1
)

y_test = tf.keras.utils.to_categorical(test_df.label, num_classes=2)
```

In [ ]:
```
labels = ["not similar", "similar"]
```

In [ ]:
```
train_df.head()
```

Out[ ]:

	split	genre	dataset	year	sid	score	sentence1	sentence2	score_class
0	train	main-captions	MSRvid	2012test	1	5.00	A plane is taking off.	An air plane is taking off.	
1	train	main-captions	MSRvid	2012test	4	3.80	A man is playing a large flute.	A man is playing a flute.	
2	train	main-captions	MSRvid	2012test	5	3.80	A man is spreading shreded cheese on a pizza.	A man is spreading shredded cheese on an uncoo...	
3	train	main-captions	MSRvid	2012test	6	2.60	Three men are playing chess.	Two men are playing chess.	n
4	train	main-captions	MSRvid	2012test	9	4.25	A man is playing the cello.	A man seated is playing the cello.	

Filtering out unnecessary column

In [ ]:
```
filtered_train = train_df[["sentence1", "sentence2", "label"]]
filtered_valid = valid_df[["sentence1", "sentence2", "label"]]
filtered_test = test_df[["sentence1", "sentence2", "label"]]
```

In [ ]:
```
filtered_train.head()
```

Out[ ]:

	sentence1	sentence2	label
0	A plane is taking off.	An air plane is taking off.	1
1	A man is playing a large flute.	A man is playing a flute.	1
2	A man is spreading shreded cheese on a pizza.	A man is spreading shredded cheese on an uncoo...	1
3	Three men are playing chess.	Two men are playing chess.	0
4	A man is playing the cello.	A man seated is playing the cello.	1

In [ ]:
```
import os

os.environ["WANDB_DISABLED"] = "true"
```

Creating a custom PyTorch dataset class for tokenizing and preparing sentence pairs for training

In [ ]:
```
import pandas as pd
import torch
from transformers import (
    BertTokenizer,
    BertForSequenceClassification,
    Trainer,
    TrainingArguments,
)

from sklearn.model_selection import train_test_split
from torch.utils.data import Dataset


class CustomDataset(Dataset):
    def __init__(self, dataframe, tokenizer, max_length=128):
        self.data = dataframe
        self.tokenizer = tokenizer
        self.max_length = max_length

    def __len__(self):
        return len(self.data)

    def __getitem__(self, idx):
        row = self.data.iloc[idx]
        encoding = self.tokenizer(
            row["sentence1"],
            row["sentence2"],
            padding="max_length",
            truncation=True,
            max_length=self.max_length,
            return_tensors="pt",
        )

        item = {key: val.squeeze(0) for key, val in encoding.items()}
        item["labels"] = torch.tensor(row["label"], dtype=torch.long)

        return item
```

In [ ]:
```
tokenizer = BertTokenizer.from_pretrained("bert-base-uncased")
train_dataset = CustomDataset(filtered_train, tokenizer)
test_dataset = CustomDataset(filtered_test, tokenizer)

model = BertForSequenceClassification.from_pretrained("bert-base-uncased"
```

```
tokenizer_config.json:   0%|          | 0.00/48.0 [00:00<?, ?B/s]
vocab.txt:   0%|          | 0.00/232k [00:00<?, ?B/s]
tokenizer.json:   0%|          | 0.00/466k [00:00<?, ?B/s]
config.json:   0%|          | 0.00/570 [00:00<?, ?B/s]
model.safetensors:   0%|          | 0.00/440M [00:00<?, ?B/s]
```

```
Some weights of BertForSequenceClassification were not initialized from th
e model checkpoint at bert-base-uncased and are newly initialized: ['class
ifier.bias', 'classifier.weight']
You should probably TRAIN this model on a down-stream task to be able to u
se it for predictions and inference.
```

Training with Hugging Face Trainer

The Hugging Face Trainer is a high-level API provided by the Hugging Face Transformers library that simplifies the training, evaluation, and fine-tuning of Transformer-based models like BERT, GPT, RoBERTa, and others.

In [ ]:
```python
# Define training arguments

training_args = TrainingArguments(
    output_dir="./results",
    num_train_epochs=3,
    per_device_train_batch_size=16,
    per_device_eval_batch_size=16,
    warmup_steps=500,
    weight_decay=0.01,
    logging_dir="./logs",
    logging_steps=10,
    evaluation_strategy="epoch",
    save_strategy="epoch",
    save_total_limit=2,
    load_best_model_at_end=True,
    report_to=["none"],  # Disable wandb
)

# Define Trainer
trainer = Trainer(
    model=model,  # The model to be trained
    args=training_args,  # Training arguments
    train_dataset=train_dataset,  # Training dataset
    eval_dataset=test_dataset,  # Evaluation dataset
    tokenizer=tokenizer,  # Tokenizer for preprocessing
)


# Train the model
trainer.train()
```

```
/opt/conda/lib/python3.10/site-packages/transformers/training_args.py:154
5: FutureWarning: `evaluation_strategy` is deprecated and will be removed
in version 4.46 of 🤗 Transformers. Use `eval_strategy` instead
  warnings.warn(
```

[1080/1080 04:18, Epoch 3/3]

Epoch	Training Loss	Validation Loss
1	0.460400	0.477510
2	0.288000	0.420245
3	0.033800	0.601781

Out[ ]:
```
TrainOutput(global_step=1080, training_loss=0.32483238599918507, metrics
={'train_runtime': 259.2662, 'train_samples_per_second': 66.522, 'train_
steps_per_second': 4.166, 'total_flos': 1134469092948480.0, 'train_los
s': 0.32483238599918507, 'epoch': 3.0})
```

Evaluating the model on the test dataset

Below you can see our evaluation metrics and the results

In [ ]:
```
# Evaluate the model
evaluation_metrics = trainer.evaluate()
print("Evaluation Metrics:", evaluation_metrics)
```

[87/87 00:05]

```
Evaluation Metrics: {'eval_loss': 0.420244961977005, 'eval_runtime': 5.962
4, 'eval_samples_per_second': 231.283, 'eval_steps_per_second': 14.591, 'e
poch': 3.0}
```

Saving Model

During the project, we decided to save the trained model after completing the training process.This approach has several practical advantages, especially when working with large pre-trained models like BERT, where training is time-intensive. Another reason for saving the model was ensuring that we preserved the model's best-performing state after severeal training processes.

With saving model we can accomplish efficiency, reusability, reproducibility, convenience.

In [ ]:
```
torch.save(model.state_dict(), "finalized_model.pth")
```

Loading Model

As part of the project, we prioritized making the trained model easily accessible for inference or further experimentation. To facilitate this, we saved the model state dictionary `(finalized_model.pth)` to a remote location `(Google Drive)` and implemented a mechanism to download and load it seamlessly.

Steps to Load the Model

1. Download the Model File:

- The code uses the `gdown` library to download the saved model `(finalized_model.pth)` from Google Drive.
- By specifying the file's unique ID `(file_id)`, the model can be directly retrieved via its shareable link.

2. Verify the Download:

- After downloading, the file size is checked using Python's os module to ensure the file was downloaded correctly.

3. Load the Model:

- The downloaded model file is loaded into the corresponding architecture using `torch.load` and `model.load_state_dict`. This ensures that the trained weights are properly restored in the model.

In [ ]:
```
import gdown

# New file ID from the link
file_id = "1F6xWrYqsGD-o8eSwIbUedLmtSoNNfhVv"
destination = "finalized_model.pth"

gdown.download(f"https://drive.google.com/uc?id={file_id}", destination,

import os

file_size = os.path.getsize(destination)
print(f"Downloaded file size: {file_size / (1024 * 1024):.2f} MB")
```

```
Downloading...
From (original): https://drive.google.com/uc?id=1F6xWrYqsGD-o8eSwIbUedLmtS
oNNfhVv
From (redirected): https://drive.google.com/uc?id=1F6xWrYqsGD-o8eSwIbUedLm
tSoNNfhVv&confirm=t&uuid=8fd50a6d-411e-4592-8aa0-13c00de0978e
To: /Users/ramazanovaaa/Documents/Duke files/IDS703 NLP/Document_Similarit
y_with_BERT/finalized_model.pth
100%|██████████| 438M/438M [01:50<00:00, 3.97MB/s]
Downloaded file size: 417.73 MB
```

In [ ]:
```
import torch

from transformers import BertTokenizer, BertForSequenceClassification

# Define the device (CPU or GPU)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

```
# Load the model architecture
model = BertForSequenceClassification.from_pretrained("bert-base-uncased"

# Load the saved state_dict and map it to the appropriate device
model.load_state_dict(torch.load("finalized_model.pth", map_location=devi

# Move the model to the selected device
model.to(device)
```

```
Some weights of BertForSequenceClassification were not initialized from th
e model checkpoint at bert-base-uncased and are newly initialized: ['class
ifier.bias', 'classifier.weight']
You should probably TRAIN this model on a down-stream task to be able to u
se it for predictions and inference.
/var/folders/rr/ktxl_hn97930l4x18gsrgn4r0000gn/T/ipykernel_87679/57795973.
py:11: FutureWarning: You are using `torch.load` with `weights_only=False`
(the current default value), which uses the default pickle module implicit
ly. It is possible to construct malicious pickle data which will execute a
rbitrary code during unpickling (See https://github.com/pytorch/pytorch/bl
ob/main/SECURITY.md#untrusted-models for more details). In a future releas
e, the default value for `weights_only` will be flipped to `True`. This li
mits the functions that could be executed during unpickling. Arbitrary obj
ects will no longer be allowed to be loaded via this mode unless they are
explicitly allowlisted by the user via `torch.serialization.add_safe_globa
ls`. We recommend you start setting `weights_only=True` for any use case w
here you don't have full control of the loaded file. Please open an issue
on GitHub for any issues related to this experimental feature.
  model.load_state_dict(torch.load('finalized_model.pth', map_location=dev
ice))
```

Out[ ]:
```
BertForSequenceClassification(
  (bert): BertModel(
    (embeddings): BertEmbeddings(
      (word_embeddings): Embedding(30522, 768, padding_idx=0)
      (position_embeddings): Embedding(512, 768)
      (token_type_embeddings): Embedding(2, 768)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (encoder): BertEncoder(
      (layer): ModuleList(
        (0-11): 12 x BertLayer(
          (attention): BertAttention(
            (self): BertSdpaSelfAttention(
              (query): Linear(in_features=768, out_features=768, bias=Tr
ue)
              (key): Linear(in_features=768, out_features=768, bias=Tru
e)
              (value): Linear(in_features=768, out_features=768, bias=Tr
ue)
              (dropout): Dropout(p=0.1, inplace=False)
            )
            (output): BertSelfOutput(
```

```
              (dense): Linear(in_features=768, out_features=768, bias=Tr
ue)
              (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affi
ne=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
          )
          (intermediate): BertIntermediate(
            (dense): Linear(in_features=768, out_features=3072, bias=Tru
e)
            (intermediate_act_fn): GELUActivation()
          )
          (output): BertOutput(
            (dense): Linear(in_features=3072, out_features=768, bias=Tru
e)
            (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine
=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
      )
    )
    (pooler): BertPooler(
      (dense): Linear(in_features=768, out_features=768, bias=True)
      (activation): Tanh()
    )
  )
  (dropout): Dropout(p=0.1, inplace=False)
  (classifier): Linear(in_features=768, out_features=2, bias=True)
)
```

Inference

The next step was to enable it to make predictions on unseen data. To achieve this, we implemented an efficient and streamlined inference pipeline, ensuring that the model could evaluate the similarity between two sentences with minimal computational overhead. And here we are tokenizing a pair of sentences, pass them through the trained model. Finally, computing probabilities and predicting the class ("**similar**" or "**not similar**").

In [ ]:
```
import torch.nn.functional as F
import torch

from transformers import (
    BertTokenizer,
    BertForSequenceClassification,
    Trainer,
    TrainingArguments,
)
```

```
def predict(sentence1, sentence2, model, tokenizer, device, max_length=12
    # Tokenize the input
    inputs = tokenizer(
        sentence1,
        sentence2,
        padding="max_length",
        truncation=True,
        max_length=max_length,
        return_tensors="pt",
    ).to(
        device
    )  # Move inputs to the same device as the model

    # Run the model
    with torch.no_grad():
        outputs = model(
            input_ids=inputs["input_ids"],
            attention_mask=inputs["attention_mask"],
        )

        logits = outputs.logits

    # Apply softmax to get probabilities
    probs = F.softmax(logits, dim=1)
    predicted_class = torch.argmax(probs).item()

    return predicted_class, probs
```

In [ ]:
```
labels = ["not similar", "similar"]
```

In [ ]:
```
device = torch.device(
    "cuda" if torch.cuda.is_available() else "cpu"
)  # using gpu if available
model.to(device)  # Move the model to the selected device
model.eval()
```

Out[ ]:
```
BertForSequenceClassification(
  (bert): BertModel(
    (embeddings): BertEmbeddings(
      (word_embeddings): Embedding(30522, 768, padding_idx=0)
      (position_embeddings): Embedding(512, 768)
      (token_type_embeddings): Embedding(2, 768)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (encoder): BertEncoder(
      (layer): ModuleList(
        (0-11): 12 x BertLayer(
          (attention): BertAttention(
            (self): BertSdpaSelfAttention(
              (query): Linear(in_features=768, out_features=768, bias=Tr
ue)
              (key): Linear(in_features=768, out_features=768, bias=Tru
e)
              (value): Linear(in_features=768, out_features=768, bias=Tr
ue)
              (dropout): Dropout(p=0.1, inplace=False)
            )
            (output): BertSelfOutput(
              (dense): Linear(in_features=768, out_features=768, bias=Tr
ue)
              (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affi
ne=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
          )
          (intermediate): BertIntermediate(
            (dense): Linear(in_features=768, out_features=3072, bias=Tru
e)
            (intermediate_act_fn): GELUActivation()
          )
          (output): BertOutput(
            (dense): Linear(in_features=3072, out_features=768, bias=Tru
e)
            (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine
=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
      )
    )
    (pooler): BertPooler(
      (dense): Linear(in_features=768, out_features=768, bias=True)
      (activation): Tanh()
    )
  )
  (dropout): Dropout(p=0.1, inplace=False)
  (classifier): Linear(in_features=768, out_features=2, bias=True)
)
```

Sentence example and prediction

Finally, the model outputs both the predicted class and the probabilities for each class. This dual output not only provides a definitive answer (e.g., "These sentences are similar") but also allows us to gauge the model's confidence in its prediction. This is particularly valuable for analyzing edge cases or uncertain predictions.

An Example in Action

Imagine comparing the sentences *"The cat is sleeping on the sofa"* and *"A cat is curled up and napping on the couch."* These sentences are tokenized, processed through the model, and classified as "similar" with a high confidence score. This result not only validates the model's training but also showcases its ability to generalize to new inputs.

In [ ]:
```
from transformers import BertTokenizer

tokenizer = BertTokenizer.from_pretrained("bert-base-uncased")
# Generate two similar sentences for testing
sentence1 = "The cat is sleeping on the sofa."
sentence2 = "A cat is curled up and napping on the couch."

# Perform inference
predicted_class, probabilities = predict(sentence1, sentence2, model, tok

# Print results
print(f"Sentence 1: {sentence1}")
print(f"Sentence 2: {sentence2}")
print(f"Predicted Class: {labels[predicted_class]}")
print(f"Class Probabilities: {probabilities}")
```

```
Sentence 1: The cat is sleeping on the sofa.
Sentence 2: A cat is curled up and napping on the couch.
Predicted Class: similar
Class Probabilities: tensor([[0.1592, 0.8408]])
```

Pros and Cons of the Model

Pros:

1. Performance: - Contextual embeddings capture nuanced relationships, leading to accurate similarity predictions.
2. Flexibility: - The pipeline can be adapted for various NLP tasks, not just similarity analysis.
3. Incremental Development: - Starting with unsupervised techniques and progressing to supervised methods ensures a robust and systematic approach.

Cons:

1. Computational Cost: - Both BERT and BART are resource-intensive, especially for large datasets.
2. Interpretability: - Transformer models are often seen as black boxes, making it hard to explain specific predictions.
3. Dependency on Data: - Performance heavily relies on the quality and quantity of training data.