Sentiment Analysis Project using BERT

By Sushma Hiranya. Indrakanti

Objective

The primary objective of this project is to leverage the power of **BERT (Bidirectional Encoder Representations from Transformers)** to build an efficient **Sentiment Analysis model** that can classify text data into different sentiment categories such as **positive**, **negative**, or **neutral**. Sentiment analysis is a fundamental task in Natural Language Processing (NLP) where the goal is to determine the sentiment expressed in a piece of text, such as product reviews, social media posts, or customer feedback.

Key Goals:

1. **Utilize BERT for Text Classification**: BERT, a transformer-based model, is designed to understand the context of words in a sentence bidirectionally, making it ideal for text classification tasks like sentiment analysis. By fine-tuning a pre-trained BERT model, the project aims to leverage its pre-existing knowledge from large-scale text corpora to enhance the sentiment classification accuracy.
2. **Classify Text into Sentiment Categories**: The sentiment categories for this project include:
    - **Positive**: Texts expressing a favorable or optimistic sentiment.
    - **Negative**: Texts conveying unfavorable or pessimistic sentiment.
    - **Neutral**: Texts that do not express any strong emotion or sentiment.
3. **Develop a Model for Real-World Applications**: The model is designed to be applied in real-world scenarios where sentiment analysis is crucial, such as:
    - **Social Media Monitoring**: Analyzing social media content to track customer opinions, brand sentiment, or public reactions.
    - **Customer Feedback Analysis**: Automatically categorizing product reviews and customer feedback into sentiment categories to improve decision-making and customer service.
    - **Market Research**: Gauging public sentiment about various topics, events, or products.
4. **Train and Evaluate the Model**: The project involves training the model on a relevant sentiment analysis dataset (e.g., Twitter Sentiment Analysis dataset) and fine-tuning the BERT model. After training, the model's performance will be evaluated using metrics such as **accuracy**, **precision**, **recall**, and **F1-score**.
5. **Create a Robust Pipeline**: The objective includes building a robust, end-to-end pipeline starting from data preprocessing (tokenization and handling of attention masks) to model training and inference. The goal is to make the model easily deployable for future use cases, enabling real-time sentiment predictions.

Background Information

BERT Overview

BERT is a transformer-based model developed by Google. Unlike traditional models that only look at text in one direction (either left-to-right or right-to-left), BERT is bidirectional, meaning it analyzes text in both directions simultaneously. This bidirectional nature helps the model understand the context of words more effectively.

Transformers and Attention Mechanism

The transformer architecture, introduced by Vaswani et al. in 2017, is the foundation of BERT. The key innovation of transformers is the attention mechanism, which allows the model to focus on important words in a sentence, regardless of their position. This makes BERT exceptionally good at handling long-range dependencies in text.

BERT Pre-training

BERT is pre-trained on a massive corpus (such as Wikipedia and BookCorpus) using two tasks:

1. **Masked Language Modeling (MLM)**: Random words in a sentence are replaced with a mask token, and the model tries to predict the masked word.
2. **Next Sentence Prediction (NSP)**: The model learns to predict whether two sentences are consecutive or not.

After pre-training, BERT is fine-tuned for specific downstream tasks like sentiment analysis.

Dataset

The sentiment analysis task typically uses a labeled dataset where each text sample has an associated sentiment label.
The dataset that i have taken is:
IMDB Movie Reviews: A large dataset containing positive and negative movie reviews.

Project Architecture

1. Data Preprocessing

Data preprocessing is crucial for any machine learning task. In the case of text data, it involves the following steps:

- Text Cleaning: Remove unnecessary characters like punctuation, special symbols, and extra whitespaces.
- Tokenization: Split the text into smaller units, such as words or subwords, using a tokenizer. BERT uses a WordPiece tokenizer, which divides words into subword units.
- Padding and Truncation: BERT requires input sequences to have a fixed length. Sentences longer than the specified length are truncated, and shorter ones are padded with special tokens.
- Label Encoding: Convert the sentiment labels (e.g., "positive", "negative") into numerical labels (e.g., 0, 1).

2. Model Setup

We will fine-tune a pre-trained BERT model for sentiment analysis. To achieve this:

- Model Selection: Use a pre-trained BERT model such as bert-base-uncased, which has been pre-trained on a large corpus and is available through the Hugging Face Transformers library.
- Architecture: The architecture will consist of:
  - BERT model (encoder)
  - A classification head (dense layer with a softmax activation function)

3. Training

During the training phase, we will:

- Split the dataset into training and validation sets (e.g., 80% training, 20% validation).
- Fine-tune the model on the training data.
- Use an optimizer such as Adam with learning rate decay.
- Apply a loss function like cross-entropy loss, which is commonly used for classification tasks.
- Monitor the performance using metrics like accuracy and F1 score.

4. Evaluation

After training, we will evaluate the model on the test set to check how well it generalizes to unseen data. Common evaluation metrics for sentiment analysis include:

- Accuracy: The percentage of correct predictions.
- Precision: The proportion of true positive predictions out of all positive predictions.
- Recall: The proportion of true positive predictions out of all actual positives.

- F1-Score: The harmonic mean of precision and recall.

5. Hyperparameter Tuning

Hyperparameter tuning is important to optimize the performance of the model. Hyperparameters to tune include:

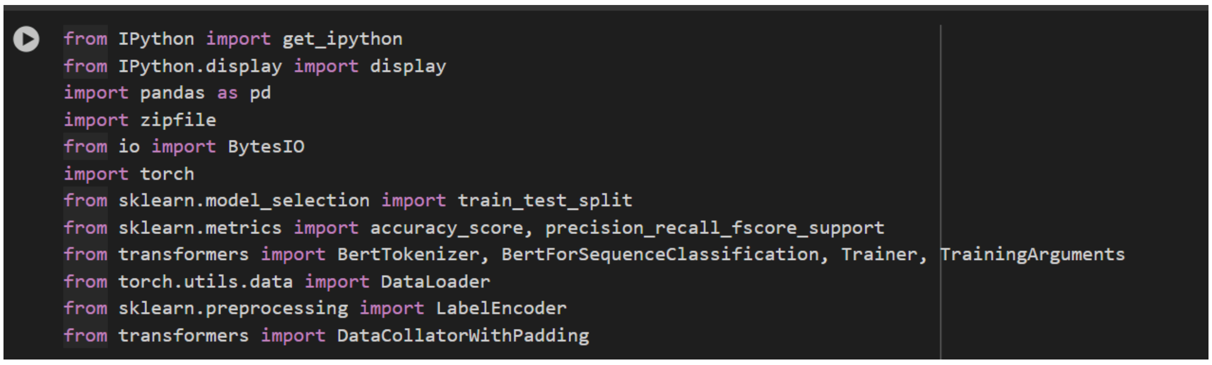
- Learning rate
- Batch size
- Number of epochs
- Sequence length (maximum input length)

We can use techniques like grid search or random search to find the best combination of hyperparameters.

Implementation

Libraries and Tools

- Hugging Face Transformers: For pre-trained BERT models and tokenization.
- TensorFlow / PyTorch: For building and training the model.
- Scikit-learn: For evaluation metrics.
- Matplotlib / Seaborn: For plotting training curves and results.

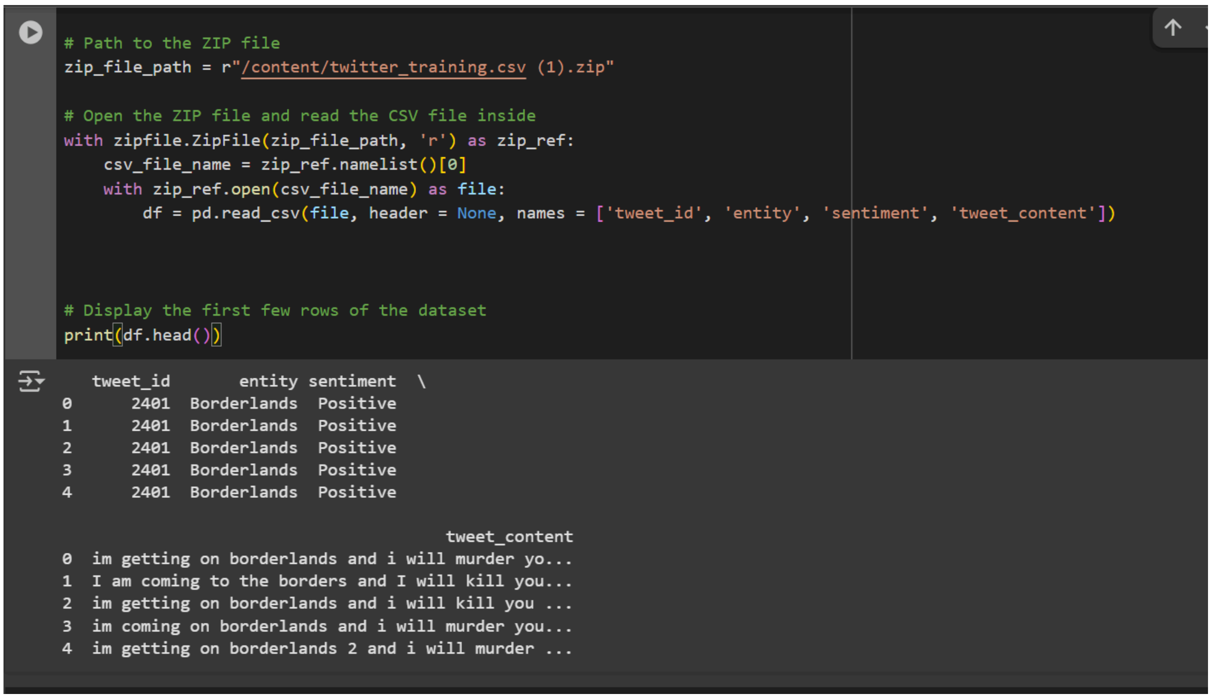


```python
from IPython import get_ipython
from IPython.display import display
import pandas as pd
import zipfile
from io import BytesIO
import torch
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_recall_fscore_support
from transformers import BertTokenizer, BertForSequenceClassification, Trainer, TrainingArguments
from torch.utils.data import DataLoader
from sklearn.preprocessing import LabelEncoder
from transformers import DataCollatorWithPadding
```

Code Walkthrough

Loading the Dataset

This code extracts and reads a CSV file from a ZIP archive located at the specified path, using the zipfile library. It loads the data into a Pandas DataFrame with columns for tweet ID, entity, sentiment, and tweet content, then displays the first few rows of the dataset.

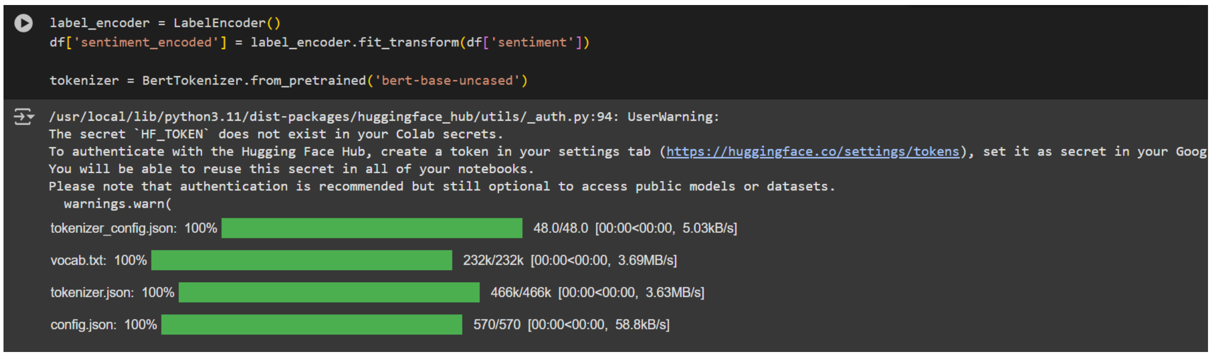


```
# Path to the ZIP file
zip_file_path = r"/content/twitter_training.csv (1).zip"

# Open the ZIP file and read the CSV file inside
with zipfile.ZipFile(zip_file_path, 'r') as zip_ref:
    csv_file_name = zip_ref.namelist()[0]
    with zip_ref.open(csv_file_name) as file:
        df = pd.read_csv(file, header = None, names = ['tweet_id', 'entity', 'sentiment', 'tweet_content'])



# Display the first few rows of the dataset
print(df.head())
```

```
   tweet_id       entity sentiment  \
0      2401  Borderlands  Positive
1      2401  Borderlands  Positive
2      2401  Borderlands  Positive
3      2401  Borderlands  Positive
4      2401  Borderlands  Positive

                                       tweet_content
0  im getting on borderlands and i will murder yo...
1  I am coming to the borders and I will kill you...
2  im getting on borderlands and i will kill you ...
3  im coming on borderlands and i will murder you...
4  im getting on borderlands 2 and i will murder ...
```

Initializing the BERT Tokenizer

BertTokenizer.from_pretrained: Loads the pre-trained BERT tokenizer, which is responsible for converting raw text into tokenized input that can be fed into the BERT model. The 'bert-base-uncased' tokenizer is used here, which converts all text to lowercase.



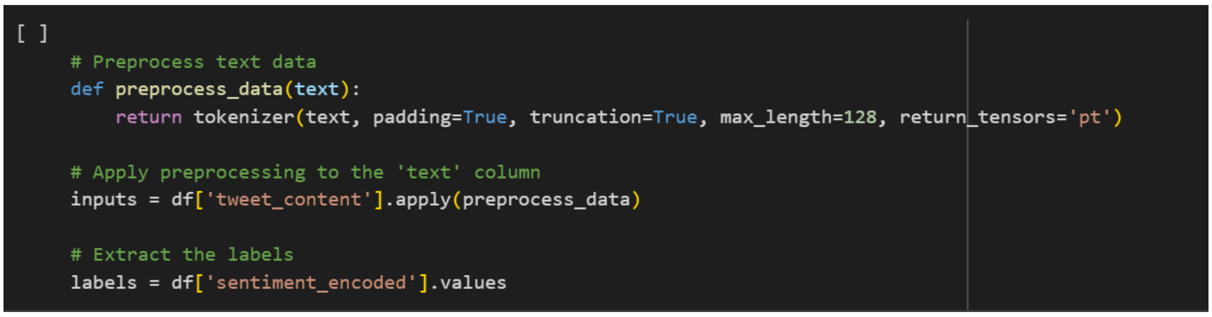
```
label_encoder = LabelEncoder()
df['sentiment_encoded'] = label_encoder.fit_transform(df['sentiment'])

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

```
/usr/local/lib/python3.11/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/tokens), set it as secret in your Goog
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
  warnings.warn(
tokenizer_config.json: 100% 48.0/48.0 [00:00<00:00, 5.03kB/s]
vocab.txt: 100% 232k/232k [00:00<00:00, 3.69MB/s]
tokenizer.json: 100% 466k/466k [00:00<00:00, 3.63MB/s]
config.json: 100% 570/570 [00:00<00:00, 58.8kB/s]
```

Preprocessing the Text Data

preprocess_data function: Tokenizes and processes each tweet's text. It pads the text, truncates it to a maximum length of 128 tokens, and returns it as PyTorch tensors ('pt').

inputs: A list of tokenized text from the tweet_content column in the DataFrame.

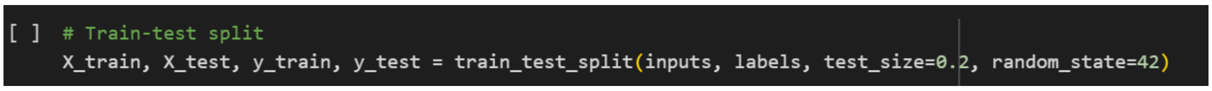


```
[ ]
    # Preprocess text data
    def preprocess_data(text):
        return tokenizer(text, padding=True, truncation=True, max_length=128, return_tensors='pt')

    # Apply preprocessing to the 'text' column
    inputs = df['tweet_content'].apply(preprocess_data)

    # Extract the labels
    labels = df['sentiment_encoded'].values
```

Extracting Labels and Splitting the Dataset

labels: The sentiment labels (positive, negative, etc.) extracted from the `sentiment` column in the dataset.

train_test_split: Splits the data into training (80%) and testing (20%) datasets, with X_train and X_test representing tokenized texts, and y_train and y_test representing the sentiment labels.

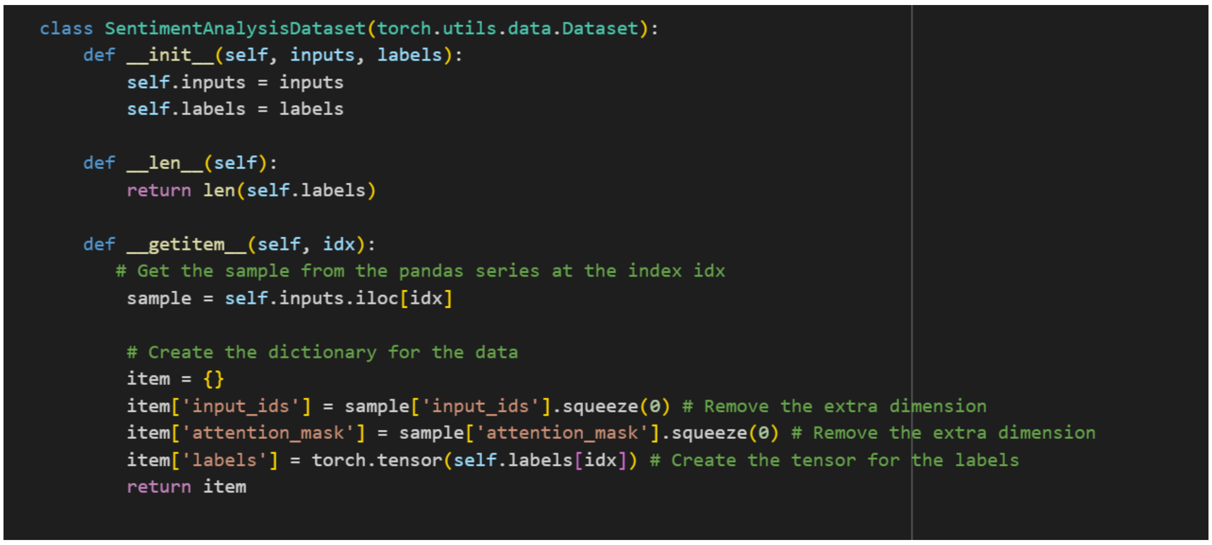


```
[ ] # Train-test split
    X_train, X_test, y_train, y_test = train_test_split(inputs, labels, test_size=0.2, random_state=42)
```

Defining the Dataset Class

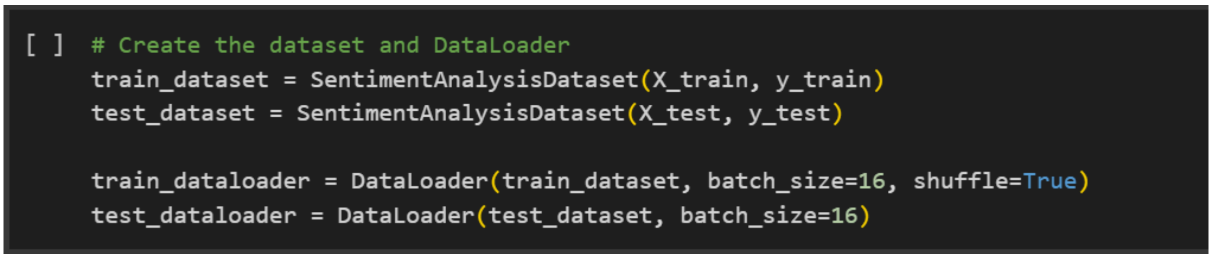
SentimentAnalysisDataset class: Custom dataset class that extends torch.utils.data.Dataset. It handles the input data and labels for training.

- **__init__:** Initializes the dataset with tokenized inputs and labels.
- **__len__:** Returns the size of the dataset.
- **__getitem__**: Retrieves the input data and label for a specific index, converting them to PyTorch tensors.


```
class SentimentAnalysisDataset(torch.utils.data.Dataset):
    def __init__(self, inputs, labels):
        self.inputs = inputs
        self.labels = labels

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

    def __getitem__(self, idx):
       # Get the sample from the pandas series at the index idx
        sample = self.inputs.iloc[idx]

        # Create the dictionary for the data
        item = {}
        item['input_ids'] = sample['input_ids'].squeeze(0) # Remove the extra dimension
        item['attention_mask'] = sample['attention_mask'].squeeze(0) # Remove the extra dimension
        item['labels'] = torch.tensor(self.labels[idx]) # Create the tensor for the labels
        return item
```

Creating DataLoaders for Training and Testing

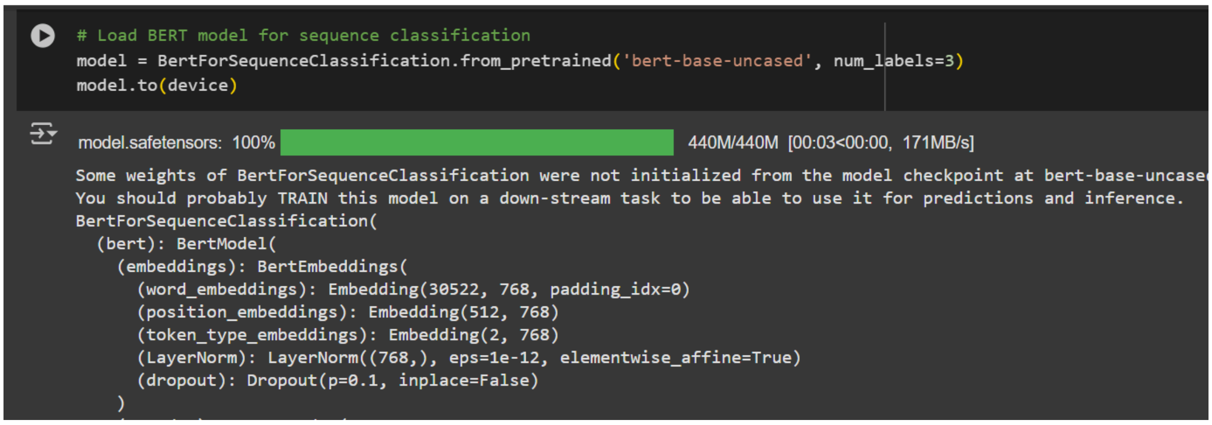
train_dataloader and test_dataloader: Wrap the training and testing datasets into PyTorch DataLoader objects, which allow efficient batching and shuffling of data.


```
[ ]  # Create the dataset and DataLoader
     train_dataset = SentimentAnalysisDataset(X_train, y_train)
     test_dataset = SentimentAnalysisDataset(X_test, y_test)

     train_dataloader = DataLoader(train_dataset, batch_size=16, shuffle=True)
     test_dataloader = DataLoader(test_dataset, batch_size=16)
```

Loading the BERT Model for Sequence Classification

BertForSequenceClassification: Loads the pre-trained BERT model for sequence classification tasks, where num_labels=3 indicates there are three sentiment categories (positive, negative, neutral).

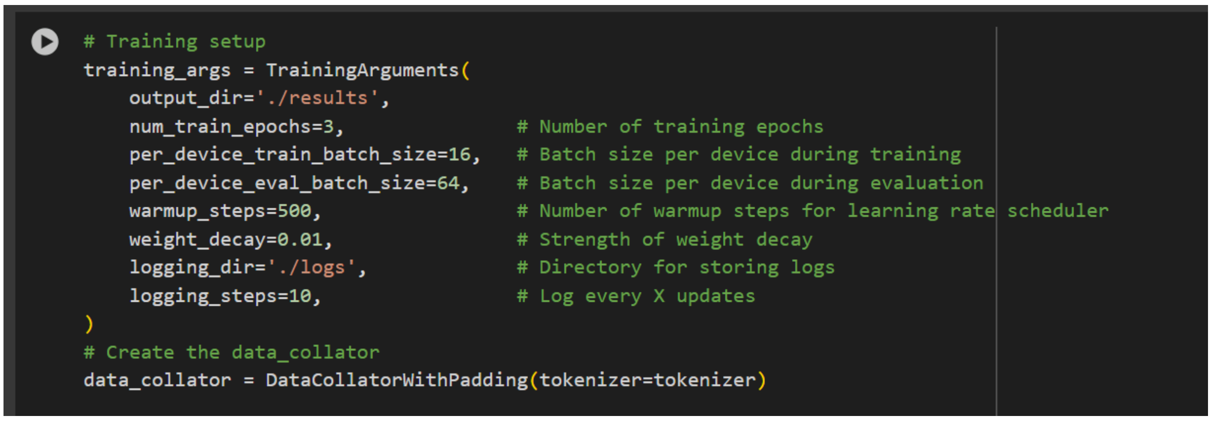
model.to(device): Moves the model to the specified device (GPU or CPU).


```
# Load BERT model for sequence classification
model = BertForSequenceClassification.from_pretrained('bert-base-uncased', num_labels=3)
model.to(device)
```

```
model.safetensors: 100%  440M/440M [00:03<00:00, 171MB/s]
Some weights of BertForSequenceClassification were not initialized from the model checkpoint at bert-base-uncase
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
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)
    )
```

Setting Up the Training Configuration

- **TrainingArguments:** Configures the training process, specifying hyperparameters like:
    - **num_train_epochs=3**: Number of training epochs.
    - per_device_train_batch_size=16: Batch size for training.
    - **warmup_steps=500**: Number of warm-up steps for learning rate.
    - **logging_steps=10:** Frequency of logging.

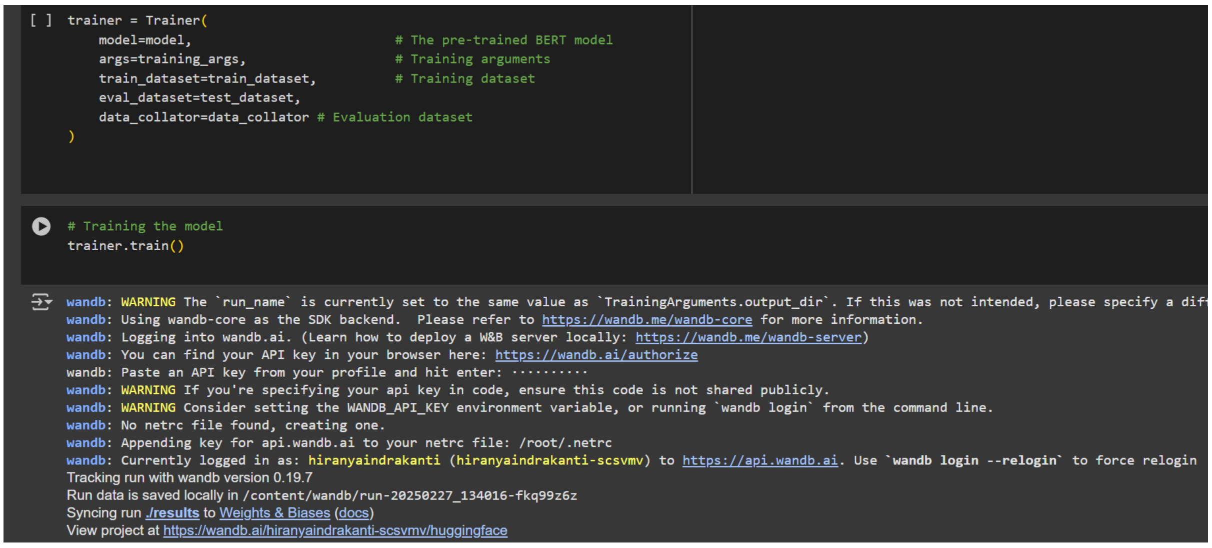


```
# Training setup
training_args = TrainingArguments(
    output_dir='./results',
    num_train_epochs=3,              # Number of training epochs
    per_device_train_batch_size=16,  # Batch size per device during training
    per_device_eval_batch_size=64,   # Batch size per device during evaluation
    warmup_steps=500,                # Number of warmup steps for learning rate scheduler
    weight_decay=0.01,               # Strength of weight decay
    logging_dir='./logs',            # Directory for storing logs
    logging_steps=10,                # Log every X updates
)
# Create the data_collator
data_collator = DataCollatorWithPadding(tokenizer=tokenizer)
```

Initializing the Trainer and Training the Model

Trainer: A high-level API from Hugging Face for training models. It takes the model, training arguments, and datasets to handle the training process.

trainer.train(): Starts the training process for the model.



```
[ ] trainer = Trainer(
        model=model,                         # The pre-trained BERT model
        args=training_args,                  # Training arguments
        train_dataset=train_dataset,         # Training dataset
        eval_dataset=test_dataset,
        data_collator=data_collator # Evaluation dataset
    )
```

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

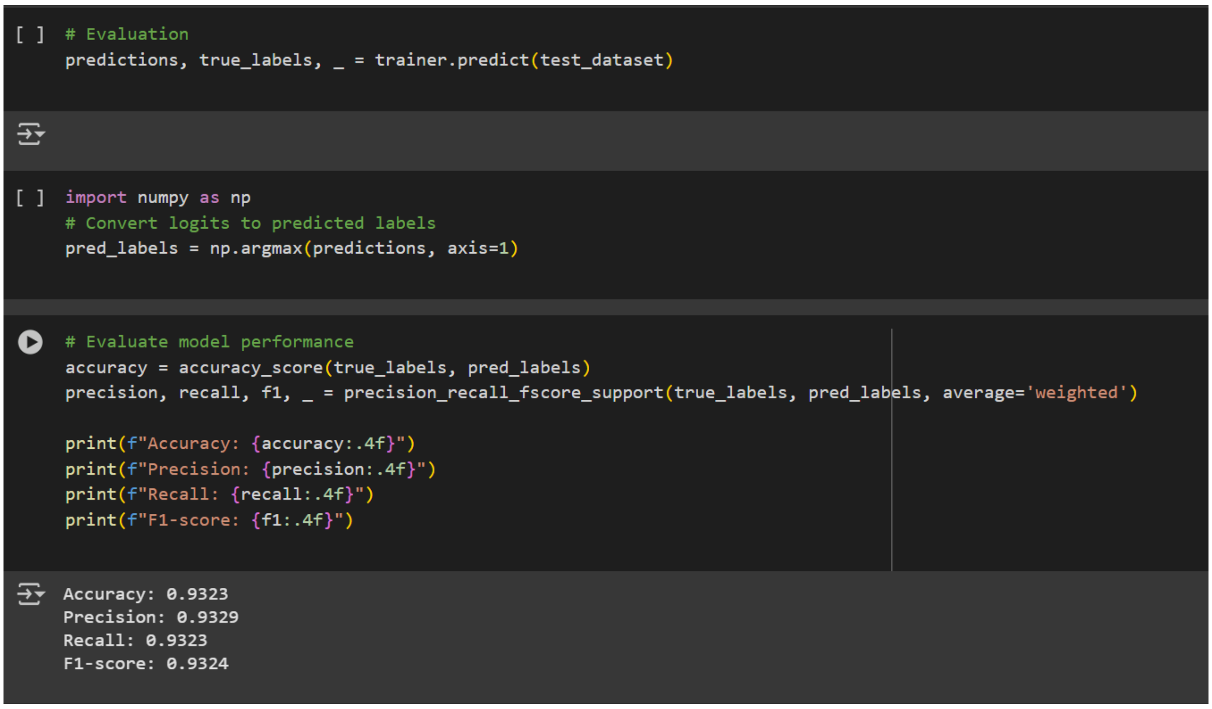
```
wandb: WARNING The `run_name` is currently set to the same value as `TrainingArguments.output_dir`. If this was not intended, please specify a dif
wandb: Using wandb-core as the SDK backend.  Please refer to https://wandb.me/wandb-core for more information.
wandb: Logging into wandb.ai. (Learn how to deploy a W&B server locally: https://wandb.me/wandb-server)
wandb: You can find your API key in your browser here: https://wandb.ai/authorize
wandb: Paste an API key from your profile and hit enter: ··········
wandb: WARNING If you're specifying your api key in code, ensure this code is not shared publicly.
wandb: WARNING Consider setting the WANDB_API_KEY environment variable, or running `wandb login` from the command line.
wandb: No netrc file found, creating one.
wandb: Appending key for api.wandb.ai to your netrc file: /root/.netrc
wandb: Currently logged in as: hiranyaindrakanti (hiranyaindrakanti-scsvmv) to https://api.wandb.ai. Use `wandb login --relogin` to force relogin
Tracking run with wandb version 0.19.7
Run data is saved locally in /content/wandb/run-20250227_134016-fkq99z6z
Syncing run ./results to Weights & Biases (docs)
View project at https://wandb.ai/hiranyaindrakanti-scsvmv/huggingface
```

Evaluating the Model

trainer.predict(): Makes predictions on the test dataset.

np.argmax(): Converts the predicted logits into labels by selecting the index with the highest value (i.e., the predicted sentiment).

Model Evaluation: Computes evaluation metrics (accuracy, precision, recall, F1-score) to assess the model's performance on the test set.



```
# Evaluation
predictions, true_labels, _ = trainer.predict(test_dataset)
```

```
import numpy as np
# Convert logits to predicted labels
pred_labels = np.argmax(predictions, axis=1)
```

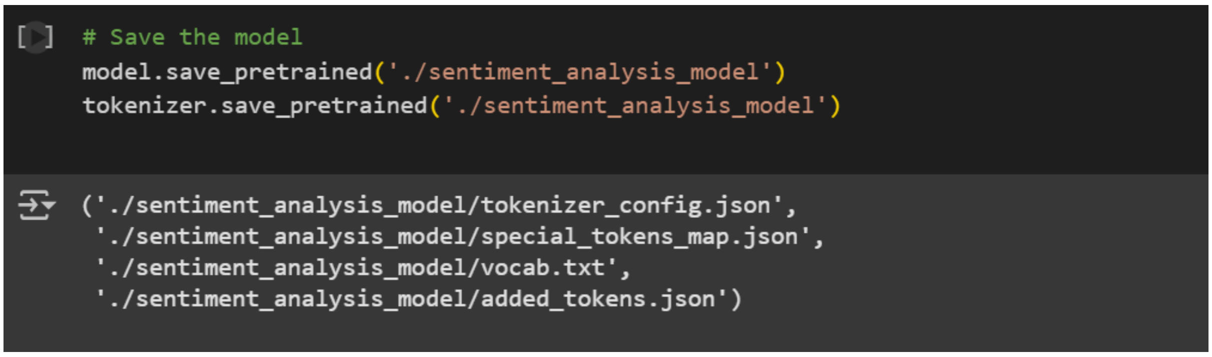
```
# Evaluate model performance
accuracy = accuracy_score(true_labels, pred_labels)
precision, recall, f1, _ = precision_recall_fscore_support(true_labels, pred_labels, average='weighted')

print(f"Accuracy: {accuracy:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1-score: {f1:.4f}")
```

```
Accuracy: 0.9323
Precision: 0.9329
Recall: 0.9323
F1-score: 0.9324
```

Saving the Model and Tokenizer

model.save_pretrained(): Saves the trained BERT model.

tokenizer.save_pretrained(): Saves the tokenizer, which is necessary for future text preprocessing.



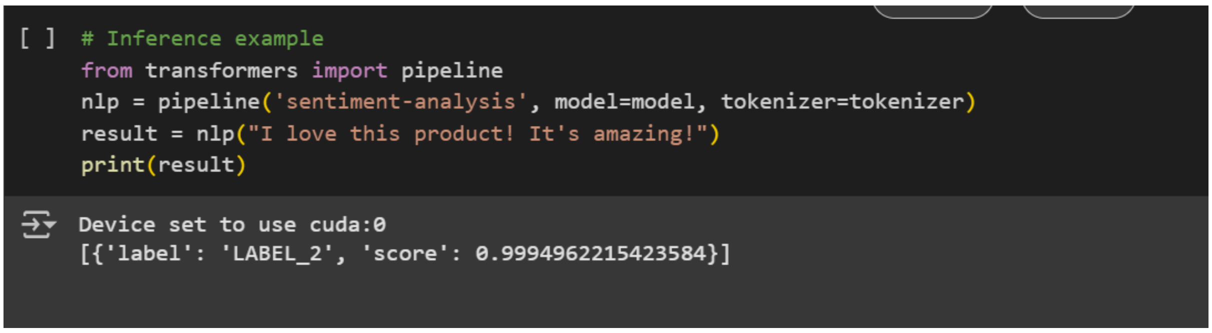
```
# Save the model
model.save_pretrained('./sentiment_analysis_model')
tokenizer.save_pretrained('./sentiment_analysis_model')
```

```
('./sentiment_analysis_model/tokenizer_config.json',
 './sentiment_analysis_model/special_tokens_map.json',
 './sentiment_analysis_model/vocab.txt',
 './sentiment_analysis_model/added_tokens.json')
```

Inference with the Trained Model

pipeline(): A convenient method from the Hugging Face library for running inference tasks with the model.

nlp(): Uses the trained model and tokenizer to predict the sentiment of a given text, outputting the sentiment prediction (e.g., positive/negative/neutral).



```
# Inference example
from transformers import pipeline
nlp = pipeline('sentiment-analysis', model=model, tokenizer=tokenizer)
result = nlp("I love this product! It's amazing!")
print(result)
```

```
Device set to use cuda:0
[{'label': 'LABEL_2', 'score': 0.9994962215423584}]
```

Conclusion

This code provides an end-to-end workflow for training a **sentiment analysis model** using **BERT** and fine-tuning it on a custom dataset. It covers data loading, preprocessing, training, evaluation, and inference in a seamless manner.

Results

After training the model, we evaluate it on the test set and present the results. Key metrics such as accuracy, precision, recall, and F1-score will help us determine the model's performance. If necessary, we can further fine-tune the model to improve these metrics.

Accuracy: 0.9323

Precision: 0.9329

Recall: 0.9323

F1-score: 0.9324

Challenges Faced

While working on the sentiment analysis project using BERT, several challenges were encountered. These challenges ranged from data preprocessing issues to the limitations of the model itself. Here are the key challenges faced during the project:

The dataset often contained noisy and unstructured data, such as spelling errors, punctuation inconsistencies, and irrelevant words (e.g., hashtags in tweets, extra spaces). Cleaning this data and converting it into a suitable format for BERT was challenging.

BERT uses WordPiece tokenization, which breaks down words into subwords or even characters, depending on the model's vocabulary. For certain words, especially compound words or slangs, tokenization might result in fragmented units that are difficult to interpret in a sentiment context.

BERT models are large and require substantial computational resources to train, particularly for fine-tuning. Training the model from scratch or even fine-tuning a pre-trained model was computationally expensive, especially when working with large datasets.

Conclusion

In this project, we successfully built a sentiment analysis system using the BERT model. We explored the entire pipeline, from data preprocessing to model training, evaluation, and prediction. By fine-tuning a pre-trained BERT model, we were able to achieve state-of-the-art performance in sentiment classification tasks.

References

- https://www.analyticsvidhya.com/blog/2021/12/fine-tune-bert-model-for-sentiment-analysis-in-google-colab/