bert-imdb-classification-5

July 9, 2024

Sentiment Analysis using BERT: Predicting Sentiment in IMDB Reviews

Author: Prashant Sundge

1 Project Index: Sentiment Analysis using BERT

1.1 1. Loading Pre-Trained BERT Model

- Initialize and load a pre-trained BERT model for fine-tuning.

1.2 2. Fine-Tuning BERT for Specific Task

- Prepare BERT for sequence classification on IMDB sentiment analysis.

1.3 3. IMDB Dataset Loading

- Load the IMDB dataset consisting of movie reviews labeled with sentiment (positive or negative).

1.4 4. Create Labels

- Prepare labels for sentiment classes based on the dataset.

1.5 5. Data Splitting: 2K Training, 100 Testing

- Split the dataset into a training set with 2000 samples and a test set with 100 samples.

1.6 6. Save the Model

- Save the fine-tuned BERT model after training.

1.7 7. Load the Model

- Load the saved BERT model for inference.

1.8 8. Testing the Model on New Data

- Tokenize new data using BERT's tokenizer.
- Convert inputs to PyTorch tensors.
- Make predictions with the loaded model.
- Extract predicted labels.

1.9 9. Display Predictions

- Show predictions made by the model on new data.

1.10 10. Model Evaluation and Metrics

- Generate a classification report showing precision, recall, and F1-score.
- Display a confusion matrix for further evaluation.

1.11 11. Create DataFrame for Reviews, Actual, and Predicted Labels

- Organize results into a DataFrame for better visualization and analysis.

1.12 12. The End

- Conclusion and summary of the project's key findings.

This project utilizes BERT, a state-of-the-art transformer model, for sentiment analysis on the IMDB dataset. The IMDB dataset consists of movie reviews labeled as positive or negative sentiments. By fine-tuning BERT for sequence classification, the model predicts sentiment labels based on the textual content of reviews. The dataset is split into a training set of 2000 samples and a test set of 100 samples for evaluation. Performance metrics such as accuracy, precision, recall, and F1-score demonstrate the model's capability to accurately classify sentiment in natural language, showcasing the application of advanced NLP techniques in understanding and analyzing textual sentiment.

2 Install Necessory Libraries

- Install Transformers and Torch libraries: `!pip install transformers torch -q`
- Install with Accelerate for enhanced performance: `!pip install accelerate -U`
- Simplified install command for all: `!pip install transformers torch accelerate -q`

[15]:
```
# !pip install transformers torch -q
```

[13]:
```
# !pip install transformers[torch] -q
```

[11]:
```
# !pip install accelerate -U

# # Step 2: Install the required libraries
# !pip install transformers torch accelerate -q
```

```
Collecting accelerate
  Downloading accelerate-0.32.1-py3-none-any.whl (314 kB)
                             314.1/314.1
kB 4.8 MB/s eta 0:00:00
Requirement already satisfied: numpy<2.0.0,>=1.17 in
/usr/local/lib/python3.10/dist-packages (from accelerate) (1.25.2)
Requirement already satisfied: packaging>=20.0 in
/usr/local/lib/python3.10/dist-packages (from accelerate) (24.1)
```

```
Requirement already satisfied: psutil in /usr/local/lib/python3.10/dist-packages
(from accelerate) (5.9.5)
Requirement already satisfied: pyyaml in /usr/local/lib/python3.10/dist-packages
(from accelerate) (6.0.1)
Requirement already satisfied: torch>=1.10.0 in /usr/local/lib/python3.10/dist-
packages (from accelerate) (2.3.0+cu121)
Requirement already satisfied: huggingface-hub in
/usr/local/lib/python3.10/dist-packages (from accelerate) (0.23.4)
Requirement already satisfied: safetensors>=0.3.1 in
/usr/local/lib/python3.10/dist-packages (from accelerate) (0.4.3)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-
packages (from torch>=1.10.0->accelerate) (3.15.4)
Requirement already satisfied: typing-extensions>=4.8.0 in
/usr/local/lib/python3.10/dist-packages (from torch>=1.10.0->accelerate)
(4.12.2)
Requirement already satisfied: sympy in /usr/local/lib/python3.10/dist-packages
(from torch>=1.10.0->accelerate) (1.12.1)
Requirement already satisfied: networkx in /usr/local/lib/python3.10/dist-
packages (from torch>=1.10.0->accelerate) (3.3)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packages
(from torch>=1.10.0->accelerate) (3.1.4)
Requirement already satisfied: fsspec in /usr/local/lib/python3.10/dist-packages
(from torch>=1.10.0->accelerate) (2023.6.0)
Collecting nvidia-cuda-nvrtc-cu12==12.1.105 (from torch>=1.10.0->accelerate)
  Using cached nvidia_cuda_nvrtc_cu12-12.1.105-py3-none-manylinux1_x86_64.whl
(23.7 MB)
Collecting nvidia-cuda-runtime-cu12==12.1.105 (from torch>=1.10.0->accelerate)
  Using cached nvidia_cuda_runtime_cu12-12.1.105-py3-none-manylinux1_x86_64.whl
(823 kB)
Collecting nvidia-cuda-cupti-cu12==12.1.105 (from torch>=1.10.0->accelerate)
  Using cached nvidia_cuda_cupti_cu12-12.1.105-py3-none-manylinux1_x86_64.whl
(14.1 MB)
Collecting nvidia-cudnn-cu12==8.9.2.26 (from torch>=1.10.0->accelerate)
  Using cached nvidia_cudnn_cu12-8.9.2.26-py3-none-manylinux1_x86_64.whl (731.7
MB)
Collecting nvidia-cublas-cu12==12.1.3.1 (from torch>=1.10.0->accelerate)
  Using cached nvidia_cublas_cu12-12.1.3.1-py3-none-manylinux1_x86_64.whl (410.6
MB)
Collecting nvidia-cufft-cu12==11.0.2.54 (from torch>=1.10.0->accelerate)
  Using cached nvidia_cufft_cu12-11.0.2.54-py3-none-manylinux1_x86_64.whl (121.6
MB)
Collecting nvidia-curand-cu12==10.3.2.106 (from torch>=1.10.0->accelerate)
  Using cached nvidia_curand_cu12-10.3.2.106-py3-none-manylinux1_x86_64.whl
(56.5 MB)
Collecting nvidia-cusolver-cu12==11.4.5.107 (from torch>=1.10.0->accelerate)
  Using cached nvidia_cusolver_cu12-11.4.5.107-py3-none-manylinux1_x86_64.whl
(124.2 MB)
Collecting nvidia-cusparse-cu12==12.1.0.106 (from torch>=1.10.0->accelerate)
```

```
  Using cached nvidia_cusparse_cu12-12.1.0.106-py3-none-manylinux1_x86_64.whl
(196.0 MB)
Collecting nvidia-nccl-cu12==2.20.5 (from torch>=1.10.0->accelerate)
  Using cached nvidia_nccl_cu12-2.20.5-py3-none-manylinux2014_x86_64.whl (176.2
MB)
Collecting nvidia-nvtx-cu12==12.1.105 (from torch>=1.10.0->accelerate)
  Using cached nvidia_nvtx_cu12-12.1.105-py3-none-manylinux1_x86_64.whl (99 kB)
Requirement already satisfied: triton==2.3.0 in /usr/local/lib/python3.10/dist-
packages (from torch>=1.10.0->accelerate) (2.3.0)
Collecting nvidia-nvjitlink-cu12 (from nvidia-cusolver-
cu12==11.4.5.107->torch>=1.10.0->accelerate)
  Downloading nvidia_nvjitlink_cu12-12.5.82-py3-none-manylinux2014_x86_64.whl
(21.3 MB)
                           21.3/21.3 MB
15.3 MB/s eta 0:00:00
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-
packages (from huggingface-hub->accelerate) (2.31.0)
Requirement already satisfied: tqdm>=4.42.1 in /usr/local/lib/python3.10/dist-
packages (from huggingface-hub->accelerate) (4.66.4)
Requirement already satisfied: MarkupSafe>=2.0 in
/usr/local/lib/python3.10/dist-packages (from jinja2->torch>=1.10.0->accelerate)
(2.1.5)
Requirement already satisfied: charset-normalizer<4,>=2 in
/usr/local/lib/python3.10/dist-packages (from requests->huggingface-
hub->accelerate) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-
packages (from requests->huggingface-hub->accelerate) (3.7)
Requirement already satisfied: urllib3<3,>=1.21.1 in
/usr/local/lib/python3.10/dist-packages (from requests->huggingface-
hub->accelerate) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.10/dist-packages (from requests->huggingface-
hub->accelerate) (2024.6.2)
Requirement already satisfied: mpmath<1.4.0,>=1.1.0 in
/usr/local/lib/python3.10/dist-packages (from sympy->torch>=1.10.0->accelerate)
(1.3.0)
Installing collected packages: nvidia-nvtx-cu12, nvidia-nvjitlink-cu12, nvidia-
nccl-cu12, nvidia-curand-cu12, nvidia-cufft-cu12, nvidia-cuda-runtime-cu12,
nvidia-cuda-nvrtc-cu12, nvidia-cuda-cupti-cu12, nvidia-cublas-cu12, nvidia-
cusparse-cu12, nvidia-cudnn-cu12, nvidia-cusolver-cu12, accelerate
Successfully installed accelerate-0.32.1 nvidia-cublas-cu12-12.1.3.1 nvidia-
cuda-cupti-cu12-12.1.105 nvidia-cuda-nvrtc-cu12-12.1.105 nvidia-cuda-runtime-
cu12-12.1.105 nvidia-cudnn-cu12-8.9.2.26 nvidia-cufft-cu12-11.0.2.54 nvidia-
curand-cu12-10.3.2.106 nvidia-cusolver-cu12-11.4.5.107 nvidia-cusparse-
cu12-12.1.0.106 nvidia-nccl-cu12-2.20.5 nvidia-nvjitlink-cu12-12.5.82 nvidia-
nvtx-cu12-12.1.105
```

3 Loading PreTrained BERT Model

- **BERT Tokenizer and Model Usage**:
  - Import `BertTokenizer` and `BertModel` from `transformers`.
  - Load pre-trained tokenizer and model using `'bert-base-uncased'`.
  - Tokenize input text with `tokenizer(text, return_tensors='pt')`.
  - Obtain embeddings using `model(**inputs).last_hidden_state`.
  - Print the shape of embeddings (`torch.Size([1, seq_length, 768])`) for BERT base model.

[1]:
```
from transformers import BertTokenizer, BertModel
# load pre-trained model and tokenizer

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


# tokenizer input text
text= "Hello, how are you ?"

inputs = tokenizer(text, return_tensors = 'pt')


# get the embeddings
outputs = model(**inputs)
last_hidden_states = outputs.last_hidden_state

print(last_hidden_states.shape)
```

```
/usr/local/lib/python3.10/dist-packages/huggingface_hub/utils/_token.py:89:
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 Google Colab
and restart your session.
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(
```

```
torch.Size([1, 8, 768])
```

4 Fine Tuning BERT for Specific Task

[2]:
```
from transformers import BertTokenizer, BertForSequenceClassification, Trainer␣
↪, TrainingArguments

import torch
```

```
from torch.utils.data import DataLoader,Dataset
```

5 IMBD DATASET LOAD

[3]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
```

[4]:
```
import pandas as pd
import zipfile
import os

# i have changed the Colab as earlier i used zip file so the # code is for that␣
 ↪but its manageable

# zip_path ="/content/drive/MyDrive/Colab Notebooks/archive/IMDB Dataset.csv"
extract_path = '/content/drive/MyDrive/Colab Notebooks/archive/IMDB Dataset.csv'

# with zipfile.ZipFile(zip_path, 'r') as zip_ref:
#     zip_ref.extractall(extract_path)

# extracted_files = os.listdir(extract_path)
# print(f"Extracted files {extracted_files}")

# dfs = []

# for file in extracted_files:
#   if file.endswith('.csv'):
#     file_path=os.path.join(extract_path, file)
#     df= pd.read_csv(file_path)
#     dfs.append(df)
```

[5]:
```
# df.head()
df = pd.read_csv(extract_path)
```

[6]:
```
df['sentiment'].value_counts()
```

[6]:
```
sentiment
positive    25000
negative    25000
Name: count, dtype: int64
```

#Create Labels

[7]:
```
def sentiments_to_labels(sentiment):
  if sentiment == "positive":
      return 1
  elif sentiment =='negative':
      return 0
  else:
      return None

df['labels'] = df['sentiment'].apply(sentiments_to_labels)

df.head()
```

[7]:

	review	sentiment	labels
0	One of the other reviewers has mentioned that …	positive	1
1	A wonderful little production. The…	positive	1
2	I thought this was a wonderful way to spend ti…	positive	1
3	Basically there's a family where a little boy …	negative	0
4	Petter Mattei's "Love in the Time of Money" is…	positive	1

6 Split data and use only 2k for training and 100 for testing

[29]:
```
df = df.drop(columns='sentiment')
```

[66]:
```
df.head()

texts = df['review'].head(2000)
labels= df['labels'].head(2000)


# testing datase created 100 for evaluation
test_reviews = list(df['review'].head(100))
test_labels =  list(df['labels'].head(100))
```

6.0.1 Dataset Preparation

- **Dataset Class**: Defined `sampleDataset` class inherits from `Dataset`, designed to handle text classification tasks using BERT.
  - **Initialization**: Accepts `texts`, `labels`, `tokenizer`, and `max_len` parameters.
  - **len Method**: Returns the length of the dataset.
  - **getitem Method**: Tokenizes each text using the tokenizer, truncates/pads to `max_len`, and returns `input_ids`, `attention_mask`, and `labels`.
- **Data Preparation**:
  - `texts` and `labels` are extracted from `df['review']` and `df['labels']` respectively.
  - `tokenizer` is initialized using `BertTokenizer.from_pretrained`.
  - `dataset` is created using `sampleDataset` with `texts`, `labels`, `tokenizer`, and `max_len=32`.

- `dataloader` is created using `DataLoader` for batching.

6.0.2 Model Training

- **Model Loading**:
  - `model` is loaded using `BertForSequenceClassification.from_pretrained` with `num_labels=2` for binary classification.
- **Training Arguments**:
  - Initial training arguments (commented out) include output directory, epochs, batch size, and logging settings.
  - Hyperparameter tuning: New `training_args` with 5 epochs, batch size 4, learning rate 5e-5, and increased logging steps.
- **Trainer Initialization**:
  - `trainer` is initialized with `model`, `training_args`, and `train_dataset`.
- **Training**:
  - `trainer.train()` initiates the training process.
- **Evaluation**:
  - `results` from `trainer.evaluate(eval_dataset=dataset)` provides evaluation metrics after training.

[91]:
```
class sampleDataset(Dataset):
  def __init__(self, texts, labels, tokenizer, max_len):
    self.texts = texts
    self.labels = labels
    self.tokenizer = tokenizer
    self.max_len = max_len

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

  def __getitem__(self, idx):
    text = self.texts[idx]
    label= self.labels[idx]
    encoding = self.tokenizer(text, truncation= True, padding='max_length',␣
 ↪max_length= self.max_len, return_tensors = 'pt')

    return {

            'input_ids': encoding['input_ids'].flatten(),
            'attention_mask': encoding['attention_mask'].flatten(),
            'labels': torch.tensor(label,dtype=torch.long)

        }

# prepare the dataset
texts = df['review'].head(2000)
labels= df['labels'].head(2000)
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
```

```
dataset = sampleDataset(texts, labels, tokenizer, max_len=32)

# create dataloader
dataloader = DataLoader(dataset, batch_size=2)

# load Model
model = BertForSequenceClassification.from_pretrained('bert-base-uncased',␣
 ↪num_labels = 2)

# # training arguments
# training_args = TrainingArguments(
#     output_dir = '.results',  # output directory
#     num_train_epochs = 1, # number of training epochs
#     per_device_train_batch_size=2, # batch size for training
#     logging_dir ='./logs',

# )
"""The above parameter are working fine
{'eval_loss': 0.5178326368331909, 'eval_runtime': 15.8858,␣
 ↪'eval_samples_per_second': 125.899, 'eval_steps_per_second': 15.737, 'epoch':
 ↪ 1.0}
"""

# Hyper Parameter Tuning
# Define training arguments with different hyperparameters
training_args = TrainingArguments(
    output_dir='./results',
    num_train_epochs=5,
    per_device_train_batch_size=4,
    learning_rate=5e-5,
    logging_dir='./logs',
    logging_steps=10,
)
"""We are trying to do hyper parameter tuning and see the results
{'eval_loss': 0.02625814639031887, 'eval_runtime': 15.9143,␣
 ↪'eval_samples_per_second': 125.673, 'eval_steps_per_second': 15.709, 'epoch':
 ↪ 5.0}

Lower Evaluation Loss: eval_loss 0.026 from epoch 5 is significantly lower than␣
 ↪eval_loss 0.51 from epoch 1.
 This suggests that the model's performance improved substantially as training␣
 ↪progressed. Therefore, eval_loss 0.026 is better than eval_loss 0.51.

"""

# trainer
trainer = Trainer(
```

```
    model = model,
    args= training_args,
    train_dataset = dataset
)

trainer.train()

# evaluate the model
results = trainer.evaluate(eval_dataset=dataset)
print(results)
```

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

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

```
{'eval_loss': 0.02625814639031887, 'eval_runtime': 15.9143,
'eval_samples_per_second': 125.673, 'eval_steps_per_second': 15.709, 'epoch':
5.0}
```

7 Save the Model

[9]:
```
model_path = "./fine_tuned_bert_model"
trainer.save_model(model_path)
```

8 Laod the Model

[10]:
```
from  transformers import BertForSequenceClassification

model= BertForSequenceClassification.from_pretrained(model_path)
```

9 Testing the Model on New Data

[67]:
```
# example of testing on New data
new_texts = test_reviews
```

10 Tokenize the new data

[68]:
```
inputs = tokenizer(new_texts, truncation= True, padding='max_length',␣
 ↪max_length=32, return_tensors = 'pt')
```

11 Convert Inputs to Pytorch

[69]:
```
# convert inputs to pytorch tensors
input_ids = inputs['input_ids']
attention_mask = inputs['attention_mask']
```

12 Make Predictions

[70]:
```
with torch.no_grad():
  outputs= model(input_ids, attention_mask=attention_mask)
```

13 extract Predicted labels

[72]:
```
predicted_labels = torch.argmax(outputs.logits, dim=1)
```

[79]:
```
predicted_labels
```

[79]:
```
tensor([1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0,
        0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0,
        1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0,
        1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1,
        0, 0, 0, 1])
```

14 display Predictions

[92]:
```
for text , label in zip(new_texts[:10], predicted_labels[:10]):
  if label == 0:
    label = "NEGATIVE"
    print(f"Text:{text[:100]} ---> Predicted Label :{label}")
  else:
    label = "POSITIVE"
    print(f"Text:{text[:100]} ---> Predicted Label :{label}")
```

```
Text:One of the other reviewers has mentioned that after watching just 1 Oz
episode you'll be hooked. The ---> Predicted Label :POSITIVE
Text:A wonderful little production. <br /><br />The filming technique is very
unassuming- very old-time-B ---> Predicted Label :POSITIVE
Text:I thought this was a wonderful way to spend time on a too hot summer
weekend, sitting in the air con ---> Predicted Label :POSITIVE
```

```
Text:Basically there's a family where a little boy (Jake) thinks there's a
zombie in his closet & his par ---> Predicted Label :NEGATIVE
Text:Petter Mattei's "Love in the Time of Money" is a visually stunning film to
watch. Mr. Mattei offers  ---> Predicted Label :POSITIVE
Text:Probably my all-time favorite movie, a story of selflessness, sacrifice and
dedication to a noble ca ---> Predicted Label :POSITIVE
Text:I sure would like to see a resurrection of a up dated Seahunt series with
the tech they have today i ---> Predicted Label :POSITIVE
Text:This show was an amazing, fresh & innovative idea in the 70's when it first
aired. The first 7 or 8  ---> Predicted Label :POSITIVE
Text:Encouraged by the positive comments about this film on here I was looking
forward to watching this f ---> Predicted Label :NEGATIVE
Text:If you like original gut wrenching laughter you will like this movie. If
you are young or old then y ---> Predicted Label :POSITIVE
```

[26]:
```
predicted_labels
```

[26]:
```
tensor([1, 1, 0])
```

15 Model Evaluation and Metrics

[82]:
```
from sklearn.metrics import classification_report, confusion_matrix

# true labels for your new_texts
true_labels = test_labels

# classification report
print(classification_report(true_labels , predicted_labels))

# confusion matrix
print(confusion_matrix(true_labels , predicted_labels))
```

```
              precision    recall  f1-score   support

           0       0.88      0.86      0.87        58
           1       0.81      0.83      0.82        42

    accuracy                           0.85       100
   macro avg       0.85      0.85      0.85       100
weighted avg       0.85      0.85      0.85       100

[[50  8]
 [ 7 35]]
```

Here are my inputs based on the classification results:

- **Accuracy**: The model achieves an accuracy of 85%, indicating that it correctly predicts the class for 85 out of every 100 instances.

- **Precision**:
  - Class 0 (negative class): Precision of 88% means that when the model predicts an instance as class 0, it is correct 88% of the time.
  - Class 1 (positive class): Precision of 81% indicates that when the model predicts an instance as class 1, it is correct 81% of the time.
- **Recall**:
  - Class 0: Recall of 86% suggests that the model correctly identifies 86% of all actual class 0 instances.
  - Class 1: Recall of 83% indicates that the model correctly identifies 83% of all actual class 1 instances.
- **F1-score**:
  - Class 0: F1-score of 87% balances precision and recall for class 0.
  - Class 1: F1-score of 82% balances precision and recall for class 1.
- **Support**:
  - Class 0: Represents 58 instances in the dataset.
  - Class 1: Represents 42 instances in the dataset.
- **Macro Average**:
  - Precision, recall, and F1-score are all 85%, computed by averaging their respective values across classes, treating each class equally.
- **Weighted Average**:
  - Precision, recall, and F1-score are all 85%, computed by averaging their respective values across classes, weighted by the number of instances for each class.

Overall, the model performs well with balanced precision and recall scores, indicating robust performance across both classes in the dataset. Adjustments and improvements can be made based on specific goals and requirements for further optimization.

[87]:
```
# create dataframe for actual and predicted values
pred_df = pd.DataFrame({
    'Text': new_texts,
    'Actual Label': test_labels,
    'Predicted Label': predicted_labels
})

# Display the DataFrame
pred_df.head() # Print the first few rows to verify
```

[87]:

	Text	Actual Label	\
0	One of the other reviewers has mentioned that …	1	
1	A wonderful little production. The…	1	
2	I thought this was a wonderful way to spend ti…	1	
3	Basically there's a family where a little boy …	0	

```
4  Petter Mattei's "Love in the Time of Money" is…             1

   Predicted Label
0                1
1                1
2                1
3                0
4                1
```

15.1 Project Summary

This project utilized BERT, a transformer-based model, to conduct sentiment analysis on the IMDB dataset. By fine-tuning BERT for sequence classification, the model accurately predicted sentiment labels (positive or negative) from movie reviews. Key findings include: - Achieved high accuracy, precision, recall, and F1-score metrics. - Demonstrated BERT's effectiveness in understanding and classifying sentiment in natural language. - Highlighted practical applications in sentiment analysis across various domains.

Overall, the project showcases the power of advanced NLP techniques in extracting meaningful insights from textual data.

[ ]: