```
"""
Lab 13.3: Transformer-Based Text Classification using BERT
Dataset: IMDb Movie Reviews
Framework: Hugging Face Transformers + PyTorch
"""

# ================================
# 1. DATA HANDLING LIBRARIES
# ================================

import pandas as pd              # For data manipulation and analysis
import numpy as np               # For numerical operations

# ================================
# 2. DATASET LOADING
# ================================

from datasets import load_dataset   # To load dataset from Hugging Face

# ================================
# 3. TRANSFORMER MODEL COMPONENTS
# ================================

from transformers import (
    AutoTokenizer,                   # For loading pretrained tokenizer
    AutoModelForSequenceClassification,  # For loading BERT classification model
    Trainer,                         # High-level training API
    TrainingArguments                # Training configuration class
)

# ================================
# 4. PYTORCH
# ================================

import torch                         # Deep learning backend

# ================================
# 5. EVALUATION METRICS
# ================================

from sklearn.metrics import (
    accuracy_score,
    precision_recall_fscore_support,
    confusion_matrix
)

# ================================
# 6. VISUALIZATION
# ================================

import matplotlib.pyplot as plt
import seaborn as sns
```

```
"""
Load IMDb dataset from Hugging Face Datasets library.
The dataset contains 50,000 movie reviews labeled as positive or negative.
"""

# Load dataset
dataset = load_dataset("imdb")

# Display dataset structure
print(dataset)

# Display one sample review
print("\nSample Record:")
print(dataset["train"][0])
```

```
/usr/local/lib/python3.12/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
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(
README.md:         7.81k/? [00:00<00:00, 237kB/s]

plain_text/train-00000-of-00001.parquet: 100%                                  21.0M/21.0M [00:00<00:00, 33.7MB/s]

plain_text/test-00000-of-00001.parquet: 100%                                   20.5M/20.5M [00:00<00:00, 65.9MB/s]

plain_text/unsupervised-00000-of-00001.p(…): 100%                              42.0M/42.0M [00:00<00:00, 49.4MB/s]

Generating train split: 100%                                  25000/25000 [00:00<00:00, 57076.89 examples/s]

Generating test split: 100%                                   25000/25000 [00:00<00:00, 74621.67 examples/s]

Generating unsupervised split: 100%                           50000/50000 [00:00<00:00, 78184.46 examples/s]

DatasetDict({
    train: Dataset({
        features: ['text', 'label'],
        num_rows: 25000
    })
    test: Dataset({
        features: ['text', 'label'],
        num_rows: 25000
    })
    unsupervised: Dataset({
        features: ['text', 'label'],
        num_rows: 50000
    })
})

Sample Record:
{'text': 'I rented I AM CURIOUS-YELLOW from my video store because of all the controversy that surrounded it when it was fir
```

```
"""
STEP 3 — Load and Explore IMDb Dataset

This section loads the IMDb movie review dataset using the Hugging Face
datasets library, converts it into a pandas DataFrame for analysis,
and performs class distribution visualization to check whether the
dataset is balanced or imbalanced.
"""

# Import required libraries
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from datasets import load_dataset


# ==============================
# Load Dataset
# ==============================

# Load IMDb dataset from Hugging Face
# The dataset contains:
#   - 25,000 training reviews
#   - 25,000 test reviews
#   - Labels: 0 (Negative), 1 (Positive)
dataset = load_dataset("imdb")


# ==============================
# Display Sample Records
# ==============================

# Display first 3 records from the training dataset
# This helps us understand:
#   - Text column name
#   - Label column name
print("First 3 Training Records:")
print(dataset["train"][:3])


# ==============================
# Convert to Pandas DataFrame
# ==============================

# Convert training split to pandas DataFrame
# Pandas allows easier data analysis and visualization
df = pd.DataFrame(dataset["train"])
```

```
# Print dataset shape (rows, columns)
# Expected: (25000, 2)
print("\nDataset Shape (Rows, Columns):", df.shape)


# ==============================
# Analyze Class Distribution
# ==============================

# Count occurrences of each class label
# 0 = Negative review
# 1 = Positive review
print("\nClass Distribution:")
print(df["label"].value_counts())


# ==============================
# Visualize Class Distribution
# ==============================

# Plot count of each label
# This helps determine if dataset is balanced or imbalanced
sns.countplot(x="label", data=df)

# Add title to plot
plt.title("Class Distribution (0 = Negative, 1 = Positive)")

# Label axes for clarity
plt.xlabel("Review Sentiment Label")
plt.ylabel("Number of Reviews")

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

```
"""
Analyze class distribution to check whether dataset is balanced.
Balanced dataset ensures unbiased model training.
"""

# Extract labels
train_labels = dataset["train"]["label"]

# Count class occurrences
unique, counts = np.unique(train_labels, return_counts=True)

# Print class distribution
for label, count in zip(unique, counts):
    print(f"Class {label}: {count} samples")
```

```
Class 0: 12500 samples
Class 1: 12500 samples
```

```
"""
Tokenization converts text into numerical tokens that the BERT model understands.
Padding ensures equal sequence length.
Truncation cuts long reviews to fixed size.
"""

# Load pre-trained BERT tokenizer
tokenizer = AutoTokenizer.from_pretrained("bert-base-uncased")


def tokenize_function(example):
    """
    Tokenizes input text using BERT tokenizer.

    Parameters:
    -----------
    example : dict
        Dictionary containing text data.

    Returns:
    --------
    dict
        Tokenized output with input_ids and attention_mask.
    """
    return tokenizer(
        example["text"],        # Text column
        padding="max_length",   # Pad to fixed length
        truncation=True,        # Truncate long texts
```

```
        max_length=128          # Maximum token length
    )


# Apply tokenization to entire dataset
tokenized_datasets = dataset.map(tokenize_function, batched=True)
```

```
config.json: 100%                                                   570/570 [00:00<00:00, 9.42kB/s]
Warning: You are sending unauthenticated requests to the HF Hub. Please set a HF_TOKEN to enable higher rate limits and fast
WARNING:huggingface_hub.utils._http:Warning: You are sending unauthenticated requests to the HF Hub. Please set a HF_TOKEN t
tokenizer_config.json: 100%                                          48.0/48.0 [00:00<00:00, 780B/s]
vocab.txt: 100%                                                   232k/232k [00:00<00:00, 3.23MB/s]
tokenizer.json: 100%                                                 466k/466k [00:00<00:00, 5.85MB/s]
Map: 100%                                                      25000/25000 [00:33<00:00, 959.31 examples/s]
Map: 100%                                                      25000/25000 [00:27<00:00, 999.42 examples/s]
Map: 100%                                                      50000/50000 [00:59<00:00, 759.45 examples/s]
```

```
"""
Rename label column to 'labels' (required by Trainer API).
Convert dataset to PyTorch tensor format.
"""

# Rename column
tokenized_datasets = tokenized_datasets.rename_column("label", "labels")

# Set dataset format for PyTorch
tokenized_datasets.set_format(
    "torch",
    columns=["input_ids", "attention_mask", "labels"]
)

print("Dataset is ready for training!")
```

```
Dataset is ready for training!
```

```
"""
Load BERT model for sequence classification.
num_labels=2 because IMDb is binary classification.
"""

model = AutoModelForSequenceClassification.from_pretrained(
    "bert-base-uncased",
    num_labels=2
)

print("Pre-trained BERT model loaded successfully!")
```

```
model.safetensors: 100%                                              440M/440M [00:03<00:00, 203MB/s]
Loading weights: 100%                                             199/199 [00:00<00:00, 306.86it/s, Materializing param=bert.pooler.dense.weight]
BertForSequenceClassification LOAD REPORT from: bert-base-uncased
Key                                         | Status     |
--------------------------------------------+------------+-
cls.predictions.transform.LayerNorm.bias    | UNEXPECTED |
cls.predictions.bias                        | UNEXPECTED |
cls.seq_relationship.bias                   | UNEXPECTED |
cls.predictions.transform.dense.weight      | UNEXPECTED |
cls.predictions.transform.dense.bias        | UNEXPECTED |
cls.predictions.transform.LayerNorm.weight  | UNEXPECTED |
cls.seq_relationship.weight                 | UNEXPECTED |
classifier.bias                             | MISSING    |
classifier.weight                           | MISSING    |

Notes:
- UNEXPECTED    :can be ignored when loading from different task/architecture; not ok if you expect identical arch.
- MISSING   :those params were newly initialized because missing from the checkpoint. Consider training on your downstream t
Pre-trained BERT model loaded successfully!
```

```
"""
Define evaluation metrics:
Accuracy, Precision, Recall, F1-score
"""

def compute_metrics(eval_pred):
    """
    Computes classification metrics.
```

```
    Parameters:
    -----------
    eval_pred : tuple
        Contains predictions and true labels.

    Returns:
    --------
    dict
        Dictionary of evaluation metrics.
    """
    logits, labels = eval_pred

    # Convert logits to predicted class
    predictions = np.argmax(logits, axis=-1)

    # Calculate metrics
    precision, recall, f1, _ = precision_recall_fscore_support(
        labels,
        predictions,
        average="binary"
    )

    acc = accuracy_score(labels, predictions)

    return {
        "accuracy": acc,
        "precision": precision,
        "recall": recall,
        "f1": f1,
    }
```

```
training_args = TrainingArguments(
    output_dir="./results",              # Save model outputs
    learning_rate=2e-5,                  # Learning rate
    per_device_train_batch_size=8,       # Training batch size
    per_device_eval_batch_size=8,        # Evaluation batch size
    num_train_epochs=2,                  # Number of training epochs
    weight_decay=0.01,                   # Regularization
    logging_dir="./logs",                # Log directory
    logging_steps=100                    # Log every 100 steps
)
```

```
`logging_dir` is deprecated and will be removed in v5.2. Please set `TENSORBOARD_LOGGING_DIR` instead.
```

```
"""
Initialize Trainer object and start training.
Trainer simplifies training loop implementation.
"""

trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=tokenized_datasets["train"],
    eval_dataset=tokenized_datasets["test"],
    compute_metrics=compute_metrics
)

# Start training
trainer.train()
```

```
/usr/local/lib/python3.12/dist-packages/torch/utils/data/dataloader.py:775: UserWarning: 'pin_memory' argument is set as tru
  super().__init__(loader)
```

[276/6250 48:14 < 17:31:44, 0.09 it/s, Epoch 0.09/2]

Step	Training Loss
100	0.597755
200	0.419463