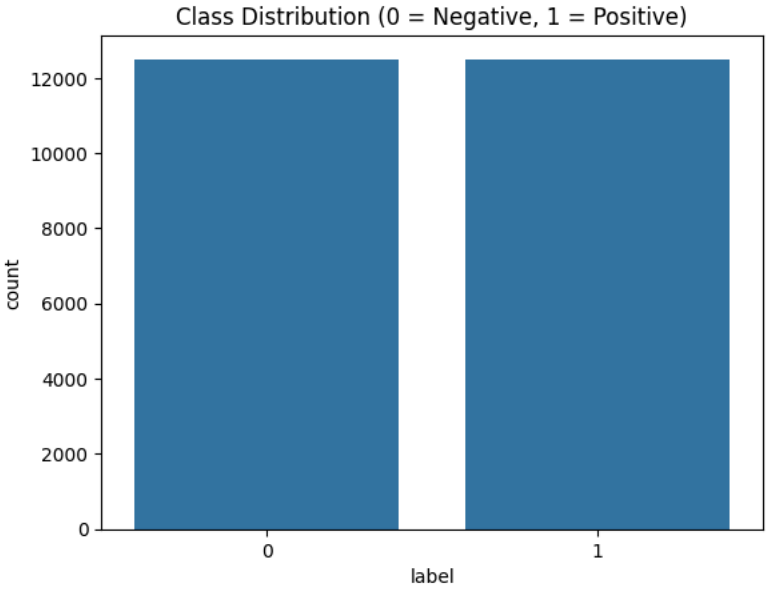
```
# Import pandas for data handling and analysis
import pandas as pd

# Import numpy for numerical operations
import numpy as np

# Import torch for deep learning tensor operations
import torch

# Import seaborn and matplotlib for visualization
import seaborn as sns
import matplotlib.pyplot as plt

# Import Hugging Face dataset loader
from datasets import load_dataset

# Import BERT tokenizer and model for sequence classification
from transformers import BertTokenizer, BertForSequenceClassification

# Import training utilities from Hugging Face
from transformers import TrainingArguments, Trainer

# Import evaluation metrics from sklearn
from sklearn.metrics import accuracy_score, precision_recall_fscore_support
from sklearn.metrics import confusion_matrix, classification_report
```

```
# Load IMDb dataset from Hugging Face (contains train and test splits)
dataset = load_dataset("imdb")

# Display first 3 records from training dataset
print(dataset["train"][:3])

# Convert training dataset to pandas DataFrame for analysis
df = pd.DataFrame(dataset["train"])

# Print dataset shape (rows, columns)
print("Dataset Shape:", df.shape)

# Display class distribution (0 = negative, 1 = positive)
print(df["label"].value_counts())

# Plot class distribution to check balance
sns.countplot(x="label", data=df)

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

# Show plot
plt.show()
```

```
/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)
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, 155kB/s]

plain_text/train-00000-of-00001.parquet: 100%                                    21.0M/21.0M [00:01<00:00, 10.5MB/s]

plain_text/test-00000-of-00001.parquet: 100%                                     20.5M/20.5M [00:00<00:00, 28.1MB/s]

plain_text/unsupervised-00000-of-00001.p(…): 100%                                    42.0M/42.0M [00:01<00:00, 39.7MB/s]

Generating train split: 100%                                   25000/25000 [00:00<00:00, 50678.25 examples/s]

Generating test split: 100%                                    25000/25000 [00:00<00:00, 72384.94 examples/s]

Generating unsupervised split: 100%                                   50000/50000 [00:00<00:00, 81674.63 examples/s]
{'text': ['I rented I AM CURIOUS-YELLOW from my video store because of all the controversy that surrounded it when it w
Dataset Shape: (25000, 2)
label
0    12500
1    12500
Name: count, dtype: int64
```



```
# Load pre-trained BERT tokenizer (uncased = lowercase text)
tokenizer = BertTokenizer.from_pretrained("bert-base-uncased")

# Define tokenization function for dataset mapping
def tokenize_function(example):
    # Convert text into input_ids and attention_mask
    return tokenizer(
        example["text"],            # Input review text
        padding="max_length",       # Pad all sequences to same length
        truncation=True,            # Truncate long reviews
        max_length=256              # Set maximum sequence length
    )

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

```
tokenizer_config.json: 100%                                   48.0/48.0 [00:00<00:00, 3.18kB/s]
Warning: You are sending unauthenticated requests to the HF Hub. Please set a HF_TOKEN to enable higher rate limits and
WARNING:huggingface_hub.utils._http:Warning: You are sending unauthenticated requests to the HF Hub. Please set a HF_TO
vocab.txt: 100%                                   232k/232k [00:00<00:00, 1.11MB/s]

tokenizer.json: 100%                                   466k/466k [00:00<00:00, 918kB/s]

Map: 100%                                   25000/25000 [00:29<00:00, 903.79 examples/s]

Map: 100%                                   25000/25000 [00:29<00:00, 811.75 examples/s]

Map: 100%                                   50000/50000 [01:09<00:00, 694.92 examples/s]
```

```
# Rename label column to 'labels' for Trainer compatibility
tokenized_dataset = tokenized_dataset.rename_column("label", "labels")

# Set dataset format to PyTorch tensors
tokenized_dataset.set_format(
    type="torch",                              # Convert to torch tensors
    columns=["input_ids", "attention_mask", "labels"]  # Required inputs for BERT
)

# Assign train dataset
train_dataset = tokenized_dataset["train"]

# Assign test dataset
test_dataset = tokenized_dataset["test"]
```

```
# Load pre-trained BERT tokenizer (uncased = lowercase text)
tokenizer = BertTokenizer.from_pretrained("bert-base-uncased")

# Define tokenization function for dataset mapping
def tokenize_function(example):
    # Convert text into input_ids and attention_mask
    return tokenizer(
        example["text"],            # Input review text
        padding="max_length",       # Pad all sequences to same length
        truncation=True,            # Truncate long reviews
        max_length=256              # Set maximum sequence length
    )

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

```
Map: 100%                                         25000/25000 [00:28<00:00, 908.38 examples/s]
```

```
# Print number of training samples
print("Train Size:", len(train_dataset))

# Print number of testing samples
print("Test Size:", len(test_dataset))
```

```
Train Size: 25000
Test Size: 25000
```

```
# Load BERT model for sequence classification with 2 output labels
model = BertForSequenceClassification.from_pretrained(
    "bert-base-uncased",   # Pre-trained BERT base model
    num_labels=2           # Binary classification (negative, positive)
)
```

```
config.json: 100%                                      570/570 [00:00<00:00, 43.6kB/s]

model.safetensors: 100%                                    440M/440M [00:03<00:00, 227MB/s]

Loading weights: 100%                                  199/199 [00:00<00:00, 408.19it/s, Materializing param=bert.pooler.dense.weight]
BertForSequenceClassification LOAD REPORT from: bert-base-uncased
Key                                         | Status     |
--------------------------------------------+------------+-
cls.predictions.transform.dense.weight      | UNEXPECTED |
cls.seq_relationship.bias                   | UNEXPECTED |
cls.predictions.transform.LayerNorm.bias    | UNEXPECTED |
cls.seq_relationship.weight                 | UNEXPECTED |
cls.predictions.transform.LayerNorm.weight  | UNEXPECTED |
cls.predictions.transform.dense.bias        | UNEXPECTED |
cls.predictions.bias                        | 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 downstr
```

```
# Define function to compute evaluation metrics
def compute_metrics(pred):
    # Extract true labels
    labels = pred.label_ids

    # Get predicted class by taking argmax of logits
    preds = pred.predictions.argmax(-1)

    # Compute precision, recall, and F1-score
    precision, recall, f1, _ = precision_recall_fscore_support(
        labels, preds, average="binary"
    )

    # Compute accuracy
    acc = accuracy_score(labels, preds)

    # Return metrics as dictionary
    return {
        "accuracy": acc,
        "precision": precision,
        "recall": recall,
        "f1": f1
    }

# Define training arguments
training_args = TrainingArguments(
    output_dir="./results",              # Directory to save model outputs
    learning_rate=2e-5,                  # Learning rate for optimizer
    per_device_train_batch_size=8,       # Batch size for training
    per_device_eval_batch_size=8,        # Batch size for evaluation
    num_train_epochs=1,                  # Number of training epochs
    # evaluation_strategy="epoch",         # Evaluate after each epoch
    save_strategy="epoch",               # Save model after each epoch
    logging_dir="./logs",                # Directory for logs
    # load_best_model_at_end=True          # Load best model based on evaluation
)
```

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

```
# Initialize Trainer with model, arguments, datasets, tokenizer, and metrics
trainer = Trainer(
    model=model,                         # BERT classification model
    args=training_args,                  # Training configuration
    train_dataset=train_dataset,         # Training data
    eval_dataset=test_dataset,           # Evaluation data
    compute_metrics=compute_metrics      # Evaluation metrics function
)

# Start model training
trainer.train()
```

```
/usr/local/lib/python3.12/dist-packages/torch/utils/data/dataloader.py:775: UserWarning: 'pin_memory' argument is set a
  super().__init__(loader)
[ 12/3125 04:41 < 24:18:05, 0.04 it/s, Epoch 0.00/1]
 Step  Training Loss
```

```
# Predict on test dataset
predictions = trainer.predict(test_dataset)

# Extract true labels
y_true = predictions.label_ids

# Extract predicted labels
y_pred = np.argmax(predictions.predictions, axis=1)

# Print classification report (precision, recall, F1, accuracy)
print(classification_report(y_true, y_pred))

# Compute confusion matrix
cm = confusion_matrix(y_true, y_pred)

# Create confusion matrix plot
plt.figure(figsize=(6,5))

# Plot heatmap of confusion matrix
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
            xticklabels=["Negative", "Positive"],
            yticklabels=["Negative", "Positive"])
```