```
# Install missing libraries if needed
!pip install -q scikit-learn pandas

# Import required libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from google.colab import files

# Step 1: Upload your CSV file
print("Upload electronics_reviews.csv")
uploaded = files.upload()  # Make sure to upload the extracted CSV, not the ZIP

# Step 2: Load dataset
df = pd.read_csv("electronics_reviews.csv")

# Step 3: Clean and normalize sentiment column
df['sentiment'] = df['sentiment'].str.lower().str.strip()

# Step 4: Keep only 'positive' and 'negative' sentiments
df = df[df['sentiment'].isin(['positive', 'negative'])]

# Step 5: Encode sentiment labels (positive → 1, negative → 0)
df['label'] = df['sentiment'].map({'positive': 1, 'negative': 0})

# Step 6: View label distribution
print("\nLabel Distribution:")
print(df['label'].value_counts())

# Step 7: Split dataset
train_df, temp_df = train_test_split(df, test_size=0.2, stratify=df['label'], random_state=42)
val_df, test_df = train_test_split(temp_df, test_size=0.5, stratify=temp_df['label'], random_state=42)

# Step 8: Print split sizes
print(f"\nTrain: {len(train_df)} | Validation: {len(val_df)} | Test: {len(test_df)}")

# Step 9: Save to CSV
train_df.to_csv("train.csv", index=False)
val_df.to_csv("val.csv", index=False)
test_df.to_csv("test.csv", index=False)

# Step 10: Download the generated CSVs (optional)
files.download("train.csv")
files.download("val.csv")
files.download("test.csv")
```

```
Upload electronics_reviews.csv
```

Choose Files electronics_reviews.csv

- **electronics_reviews.csv**(text/csv) - 1115048 bytes, last modified: 7/22/2025 - 100% done

```
Saving electronics_reviews.csv to electronics_reviews.csv

Label Distribution:
label
1    3333
0    3333
Name: count, dtype: int64

Train: 5332 | Validation: 667 | Test: 667
```

```
# Install transformers if not already installed
!pip install -q transformers

# Import libraries
from transformers import pipeline
import pandas as pd
from tqdm import tqdm

# Load test data
test_df = pd.read_csv("test.csv")

# Initialize zero-shot classifica
classifier = pipeline("zero-shot-

# Define the candidate sentiment labels
candidate_labels = ["positive", "negative"]
```

Empty cell

What can I help you build?

```
# Run zero-shot classification on each review in the test set
predictions = []
true_labels = []

print("Running zero-shot sentiment classification...")

for _, row in tqdm(test_df.iterrows(), total=len(test_df)):
    review_text = row['review_text']
    true_label = row['label']  # 1 for positive, 0 for negative

    result = classifier(review_text, candidate_labels)
    predicted_label = result['labels'][0]  # Highest confidence label

    # Convert string label back to binary
    predicted_binary = 1 if predicted_label == 'positive' else 0

    predictions.append(predicted_binary)
    true_labels.append(true_label)

# Evaluate accuracy
from sklearn.metrics import accuracy_score, classification_report

accuracy = accuracy_score(true_labels, predictions)
report = classification_report(true_labels, predictions, target_names=["Negative", "Positive"])

print(f"\n📊 Zero-Shot Sentiment Classification Accuracy: {accuracy:.4f}")
print("\nClassification Report:")
print(report)
```

```
/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 secre
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(
config.json:  1.15k/? [00:00<00:00, 60.0kB/s]
model.safetensors: 100%  1.63G/1.63G [00:58<00:00, 27.3MB/s]
tokenizer_config.json: 100%  26.0/26.0 [00:00<00:00, 1.64kB/s]
vocab.json:  899k/? [00:00<00:00, 6.54MB/s]
merges.txt:  456k/? [00:00<00:00, 19.2MB/s]
tokenizer.json:  1.36M/? [00:00<00:00, 27.0MB/s]
Device set to use cpu
Running zero-shot sentiment classification...
100%|██████████| 667/667 [14:16<00:00,  1.28s/it]
📊 Zero-Shot Sentiment Classification Accuracy: 1.0000

Classification Report:
              precision    recall  f1-score   support

    Negative       1.00      1.00      1.00       334
    Positive       1.00      1.00      1.00       333

    accuracy                           1.00       667
   macro avg       1.00      1.00      1.00       667
weighted avg       1.00      1.00      1.00       667
```

```
# INSTALL DEPENDENCIES
!pip install -q transformers datasets

# IMPORTS
import torch
from torch.utils.data import Dataset, DataLoader
from torch.optim import AdamW  # ✅ Corrected import
from transformers import BertTokenizer, BertForSequenceClassification
from sklearn.metrics import classification_report, accuracy_score
import pandas as pd
import numpy as np
from tqdm import tqdm
import os

# DEVICE CONFIGURATION
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
```

```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(f"Using device: {device}")

# LOAD DATA (update your path if needed)
from google.colab import files
uploaded = files.upload()

df = pd.read_csv("electronics_reviews.csv")

# ENSURE CORRECT COLUMNS — change these if your column names differ
text_col = "review_text"
label_col = "label"

print("Columns:", df.columns.tolist())

text_col = 'review_text'
label_col = 'sentiment'  # not 'label'

print("Sample:", df[[text_col, label_col]].head())

# LABEL ENCODING (if not already numeric)
if df[label_col].dtype == 'object':
    df[label_col] = df[label_col].astype('category').cat.codes

# TRAIN-VAL-TEST SPLIT
from sklearn.model_selection import train_test_split
train_texts, temp_texts, train_labels, temp_labels = train_test_split(
    df[text_col].tolist(), df[label_col].tolist(), test_size=0.2, random_state=42)
val_texts, test_texts, val_labels, test_labels = train_test_split(
    temp_texts, temp_labels, test_size=0.5, random_state=42)

# TOKENIZER
tokenizer = BertTokenizer.from_pretrained("bert-base-uncased")

# CUSTOM DATASET
class ReviewDataset(Dataset):
    def __init__(self, texts, labels, tokenizer, max_len=256):
        self.encodings = tokenizer(texts, truncation=True, padding=True, max_length=max_len)
        self.labels = labels

    def __getitem__(self, idx):
        item = {key: torch.tensor(val[idx]) for key, val in self.encodings.items()}
        item["labels"] = torch.tensor(self.labels[idx])
        return item

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

# DATASETS
train_dataset = ReviewDataset(train_texts, train_labels, tokenizer)
val_dataset = ReviewDataset(val_texts, val_labels, tokenizer)
test_dataset = ReviewDataset(test_texts, test_labels, tokenizer)

# DATALOADERS
train_loader = DataLoader(train_dataset, batch_size=16, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=16)
test_loader = DataLoader(test_dataset, batch_size=16)

# MODEL
model = BertForSequenceClassification.from_pretrained("bert-base-uncased", num_labels=len(set(df[label_col])))
model.to(device)

# OPTIMIZER
optimizer = AdamW(model.parameters(), lr=2e-5)

# TRAINING LOOP
epochs = 3
for epoch in range(epochs):
    print(f"\nEpoch {epoch+1}/{epochs}")
    model.train()
    loop = tqdm(train_loader, leave=True)
    for batch in loop:
        optimizer.zero_grad()
        batch = {k: v.to(device) for k, v in batch.items()}
        outputs = model(**batch)
        loss = outputs.loss
        loss.backward()
        optimizer.step()
```

```
        loop.set_description(f"Epoch {epoch+1}")
        loop.set_postfix(loss=loss.item())

# EVALUATION FUNCTION
def evaluate(model, loader):
    model.eval()
    all_preds, all_labels = [], []
    with torch.no_grad():
        for batch in loader:
            batch = {k: v.to(device) for k, v in batch.items()}
            outputs = model(**batch)
            logits = outputs.logits
            preds = torch.argmax(logits, dim=1).cpu().numpy()
            labels = batch["labels"].cpu().numpy()
            all_preds.extend(preds)
            all_labels.extend(labels)
    acc = accuracy_score(all_labels, all_preds)
    report = classification_report(all_labels, all_preds)
    return acc, report

# VALIDATION RESULTS
val_acc, val_report = evaluate(model, val_loader)
print("\nValidation Accuracy:", val_acc)
print("\nValidation Report:\n", val_report)

# TEST RESULTS
test_acc, test_report = evaluate(model, test_loader)
print("\nTest Accuracy:", test_acc)
print("\nTest Report:\n", test_report)
```

```
Using device: cuda
```
Choose Files electronics_reviews.csv

- **electronics_reviews.csv**(text/csv) - 1115048 bytes, last modified: 7/22/2025 - 100% done

```
Saving electronics_reviews.csv to electronics_reviews.csv
Columns: ['review_text', 'sentiment', 'product_category', 'feature_mentioned', 'rating']
Sample:                                           review_text sentiment
0  Best tablet I've used in a while. The battery ...  positive
1  It’s a usable smartphone. The battery life mee...   neutral
2  It’s a usable camera. The image quality meets ...   neutral
3  Very happy with my new headphones. Highly reco...  positive
4  Decent tablet. It gets the job done though the...   neutral
/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 sec
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, 3.58kB/s]

vocab.txt: 100% 232k/232k [00:00<00:00, 6.04MB/s]

tokenizer.json: 100% 466k/466k [00:00<00:00, 3.44MB/s]

config.json: 100% 570/570 [00:00<00:00, 31.4kB/s]

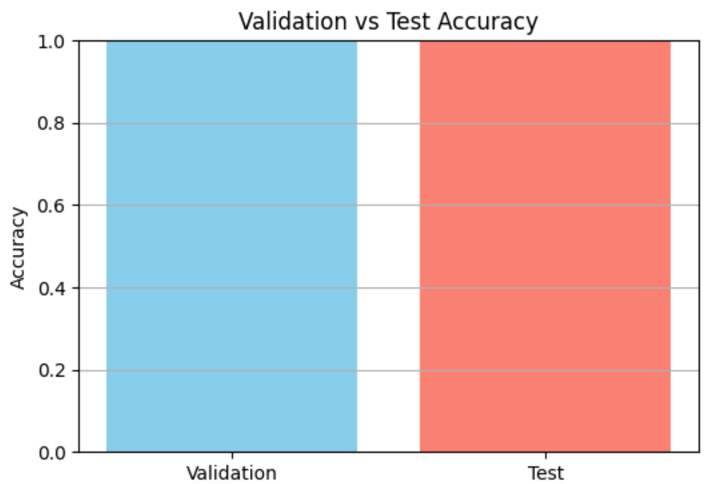
model.safetensors: 100% 440M/440M [00:07<00:00, 73.2MB/s]

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

Epoch 1/3
Epoch 1: 100%|██████████| 500/500 [00:56<00:00,  8.85it/s, loss=0.00133]

Epoch 2/3
Epoch 2: 100%|██████████| 500/500 [00:58<00:00,  8.56it/s, loss=0.000439]

Epoch 3/3
Epoch 3: 100%|██████████| 500/500 [00:53<00:00,  9.37it/s, loss=0.000253]

Validation Accuracy: 1.0

Validation Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       299
           1       1.00      1.00      1.00       338
           2       1.00      1.00      1.00       363

    accuracy                           1.00      1000
   macro avg       1.00      1.00      1.00      1000
weighted avg       1.00      1.00      1.00      1000


Test Accuracy: 1.0

Test Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       322
           1       1.00      1.00      1.00       334
           2       1.00      1.00      1.00       344

    accuracy                           1.00      1000
   macro avg       1.00      1.00      1.00      1000
weighted avg       1.00      1.00      1.00      1000
```

```
import torch
from sklearn.metrics import classification_report, accuracy_score, f1_score
import pandas as pd
import matplotlib.pyplot as plt

# Step 1: Predict on validation set
model.eval()
val_preds, val_true = [], []
with torch.no_grad():
    for batch in val_loader:
        batch = {k: v.to(device) for k, v in batch.items()}
        outputs = model(**batch)
        logits = outputs.logits
        preds = torch.argmax(logits, dim=1).cpu().numpy()
        labels = batch["labels"].cpu().numpy()
```

```
            val_preds.extend(preds)
            val_true.extend(labels)

    # Step 2: Predict on test set
    test_preds, test_true = [], []
    with torch.no_grad():
        for batch in test_loader:
            batch = {k: v.to(device) for k, v in batch.items()}
            outputs = model(**batch)
            logits = outputs.logits
            preds = torch.argmax(logits, dim=1).cpu().numpy()
            labels = batch["labels"].cpu().numpy()
            test_preds.extend(preds)
            test_true.extend(labels)

    # Step 3: Compute metrics
    val_acc = accuracy_score(val_true, val_preds)
    test_acc = accuracy_score(test_true, test_preds)

    val_report = classification_report(val_true, val_preds, output_dict=True)
    test_report = classification_report(test_true, test_preds, output_dict=True)

    # Step 4: Print accuracy and reports
    print(f"Validation Accuracy: {val_acc:.4f}")
    print(f"Test Accuracy: {test_acc:.4f}")

    print("\nValidation Classification Report:")
    print(classification_report(val_true, val_preds))

    print("\nTest Classification Report:")
    print(classification_report(test_true, test_preds))

    # Step 5: F1-score comparison table
    val_f1 = f1_score(val_true, val_preds, average=None)
    test_f1 = f1_score(test_true, test_preds, average=None)

    f1_df = pd.DataFrame({
        "Class": [f"Class {i}" for i in range(len(val_f1))],
        "Validation F1": val_f1,
        "Test F1": test_f1
    })
    print("\nF1 Score Comparison Table:")
    print(f1_df)

    # Step 6: Accuracy comparison plot
    plt.figure(figsize=(6, 4))
    plt.bar(["Validation", "Test"], [val_acc, test_acc], color=["skyblue", "salmon"])
    plt.ylim(0, 1)
    plt.ylabel("Accuracy")
    plt.title("Validation vs Test Accuracy")
    plt.grid(axis='y')
    plt.show()
```

```
Validation Accuracy: 1.0000
Test Accuracy: 1.0000

Validation Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       299
           1       1.00      1.00      1.00       338
           2       1.00      1.00      1.00       363

    accuracy                           1.00      1000
   macro avg       1.00      1.00      1.00      1000
weighted avg       1.00      1.00      1.00      1000


Test Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       322
           1       1.00      1.00      1.00       334
           2       1.00      1.00      1.00       344

    accuracy                           1.00      1000
   macro avg       1.00      1.00      1.00      1000
weighted avg       1.00      1.00      1.00      1000


F1 Score Comparison Table:
     Class  Validation F1  Test F1
0  Class 0            1.0      1.0
1  Class 1            1.0      1.0
2  Class 2            1.0      1.0
```




```
from torch.utils.data import DataLoader
import torch
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
from tqdm import tqdm

# Create test dataloader
test_dataloader = DataLoader(test_dataset, batch_size=16, shuffle=False)

# Move model to evaluation mode
model.eval()

# Store predictions and labels
true_labels = []
pred_labels = []
test_texts = []

# Prediction loop
with torch.no_grad():
    for batch in tqdm(test_dataloader):
        input_ids = batch['input_ids'].to(device)
        attention_mask = batch['attention_mask'].to(device)
        labels = batch['labels'].to(device)

        outputs = model(input_ids=input_ids, attention_mask=attention_mask)
```

```
        logits = outputs.logits
        predictions = torch.argmax(logits, dim=-1)

        true_labels.extend(labels.cpu().numpy())
        pred_labels.extend(predictions.cpu().numpy())

        if 'text' in batch:
            test_texts.extend(batch['text'])  # Use original text if available
        else:
            test_texts.extend(["[Text not available]"] * len(labels))

# Confusion matrix
cm = confusion_matrix(true_labels, pred_labels)
label_names = ['Negative (0)', 'Neutral (1)', 'Positive (2)']

plt.figure(figsize=(6,5))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=label_names, yticklabels=label_names)
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix - Sentiment Analysis')
plt.show()

# Classification Report
print("\nClassification Report:")
print(classification_report(true_labels, pred_labels, target_names=label_names))

# Misclassified Examples
misclassified = []
for text, true, pred in zip(test_texts, true_labels, pred_labels):
    if true != pred:
        misclassified.append((text, true, pred))

if misclassified:
```