```
import numpy as np
import pandas as pd
import random
from tqdm import tqdm
from sklearn.metrics import f1_score, accuracy_score
from sklearn.model_selection import train_test_split
from transformers import (
    BertTokenizer,
    AutoModelForSequenceClassification,
    AdamW,
    get_linear_schedule_with_warmup,
)
import torch
from torch.utils.data import DataLoader, RandomSampler, SequentialSampler, TensorDataset
```

```
from google.colab import files

uploaded = files.upload()

input_file = "Sentences_AllAgree.txt"
output_file = "Sentences_AllAgree.csv"

data = []
with open(input_file, "r", encoding="latin-1") as file:
    for line in file:
        if "@" in line:
            sentence, sentiment = line.rsplit("@", 1)
            data.append({"NewsHeadline": sentence.strip(), "sentiment": sentiment.strip()})

df = pd.DataFrame(data)
df.to_csv(output_file, index=False, encoding="utf-8")

print(f"File saved to {output_file}")
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving Sentences_AllAgree.txt to Sentences_AllAgree (1).txt
```

```
input_file = "Sentences_AllAgree.csv"
financial_data = pd.read_csv(input_file)
```

```
def encode_sentiments_values(df):
    possible_sentiments = df.sentiment.unique()
    sentiment_dict = {}

    for index, possible_sentiment in enumerate(possible_sentiments):
        sentiment_dict[possible_sentiment] = index

    df["label"] = df.sentiment.replace(sentiment_dict)

    return df, sentiment_dict

financial_data, sentiment_dict = encode_sentiments_values(financial_data)
```

```
print("Class distribution before adjustment:")
print(financial_data["label"].value_counts())

if (financial_data['label'].value_counts() < 2).any():
    financial_data = financial_data[
        financial_data['label'].map(financial_data['label'].value_counts()) > 1
    ]
    print("Adjusted class distribution:")
    print(financial_data["label"].value_counts())
```

```
# Split the data into training and validation sets
X_train, X_val, y_train, y_val = train_test_split(
    financial_data.index.values,
    financial_data.label.values,
    test_size=0.20,
    random_state=2022,
    stratify=financial_data.label.values,
)



# Get the BERT Tokenizer
tokenizer = BertTokenizer.from_pretrained("bert-base-uncased", do_lower_case=True)



# Encode the Training and Validation Data
encoded_data_train = tokenizer.batch_encode_plus(
    financial_data.loc[X_train, "NewsHeadline"].values,
    return_tensors="pt",
    add_special_tokens=True,
    return_attention_mask=True,
    pad_to_max_length=True,
    max_length=150,
)

encoded_data_val = tokenizer.batch_encode_plus(
    financial_data.loc[X_val, "NewsHeadline"].values,
    return_tensors="pt",
    add_special_tokens=True,
    return_attention_mask=True,
    pad_to_max_length=True,
    max_length=150,
)

# Prepare input tensors
input_ids_train = encoded_data_train["input_ids"]
attention_masks_train = encoded_data_train["attention_mask"]
labels_train = torch.tensor(y_train)

input_ids_val = encoded_data_val["input_ids"]
attention_masks_val = encoded_data_val["attention_mask"]
labels_val = torch.tensor(y_val)

# Create datasets
dataset_train = TensorDataset(input_ids_train, attention_masks_train, labels_train)
dataset_val = TensorDataset(input_ids_val, attention_masks_val, labels_val)



# DataLoaders
batch_size = 32
dataloader_train = DataLoader(
    dataset_train, sampler=RandomSampler(dataset_train), batch_size=batch_size
)
dataloader_validation = DataLoader(
    dataset_val, sampler=SequentialSampler(dataset_val), batch_size=batch_size
)



# Load Pre-trained Model
model = AutoModelForSequenceClassification.from_pretrained(
    "dogruermikail/bert-fine-tuned-stock-sentiment-uncased", num_labels=len(sentiment_dict)
)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)



# Optimizer and Scheduler
optimizer = AdamW(model.parameters(), lr=5e-5, eps=1e-8)
epochs = 3
scheduler = get_linear_schedule_with_warmup(
    optimizer, num_warmup_steps=0, num_training_steps=len(dataloader_train) * epochs
)



# Evaluation Function
def evaluate(dataloader_val):
    model.eval()
    loss_val_total = 0
    predictions, true_vals = [], []

    for batch in dataloader_val:
        batch = tuple(b.to(device) for b in batch)
        inputs = {"input_ids": batch[0], "attention_mask": batch[1], "labels": batch[2]}

        with torch.no_grad():
            outputs = model(**inputs)

        loss = outputs[0]
        logits = outputs[1]
        loss_val_total += loss.item()

        logits = logits.detach().cpu().numpy()
        label_ids = inputs["labels"].cpu().numpy()
        predictions.append(logits)
        true_vals.append(label_ids)

    loss_val_avg = loss_val_total / len(dataloader_val)

    predictions = np.concatenate(predictions, axis=0)
    true_vals = np.concatenate(true_vals, axis=0)
    return loss_val_avg, predictions, true_vals



# Training Loop
for epoch in tqdm(range(1, epochs + 1)):
    model.train()
    loss_train_total = 0
    progress_bar = tqdm(dataloader_train, desc=f"Epoch {epoch}", leave=False)

    for batch in progress_bar:
        model.zero_grad()
        batch = tuple(b.to(device) for b in batch)
        inputs = {"input_ids": batch[0], "attention_mask": batch[1], "labels": batch[2]}
        outputs = model(**inputs)
        loss = outputs[0]
        loss_train_total += loss.item()
        loss.backward()
        torch.nn.utils.clip_grad_norm_(model.parameters(), 1.0)
        optimizer.step()
        scheduler.step()
        progress_bar.set_postfix({"training_loss": loss.item()})

    torch.save(model.state_dict(), f"finetuned_BERT_epoch_{epoch}.model")
    print(f"\nEpoch {epoch}")
    print(f"Training loss: {loss_train_total / len(dataloader_train)}")

    val_loss, predictions, true_vals = evaluate(dataloader_validation)
    val_f1 = f1_score(np.argmax(predictions, axis=1), true_vals, average="weighted")
    print(f"Validation loss: {val_loss}")
    print(f"F1 Score (Weighted): {val_f1}")
```

```
<ipython-input-7-218e9b67e782>:29: FutureWarning: Downcasting behavior in `replace` is deprecated and will be removed in
  df["label"] = df.sentiment.replace(sentiment_dict)
Class distribution before adjustment:
label
0    1391
1     570
2     303
Name: count, dtype: int64
/usr/local/lib/python3.10/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(
tokenizer_config.json: 100%    48.0/48.0 [00:00<00:00, 1.34kB/s]
vocab.txt: 100%    232k/232k [00:00<00:00, 3.64MB/s]
tokenizer.json: 100%    466k/466k [00:00<00:00, 9.00MB/s]
config.json: 100%    570/570 [00:00<00:00, 22.0kB/s]
/usr/local/lib/python3.10/dist-packages/transformers/tokenization_utils_base.py:2834: FutureWarning: The `pad_to_max_len
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/transformers/tokenization_utils_base.py:2834: FutureWarning: The `pad_to_max_len
  warnings.warn(
config.json: 100%    758/758 [00:00<00:00, 49.2kB/s]
pytorch_model.bin: 100%    438M/438M [00:03<00:00, 174MB/s]
/usr/local/lib/python3.10/dist-packages/transformers/optimization.py:591: FutureWarning: This implementation of AdamW is
  warnings.warn(
  0%|          | 0/3 [00:00<?, ?it/s]
Epoch 1:   0%|          | 0/57 [00:00<?, ?it/s]
Epoch 1:   0%|          | 0/57 [00:59<?, ?it/s, training_loss=3.2]
Epoch 1:   2%|▏         | 1/57 [00:59<55:52, 59.86s/it, training_loss=3.2]
Epoch 1:   2%|▏         | 1/57 [01:58<55:52, 59.86s/it, training_loss=2.15]
Epoch 1:   4%|▎         | 2/57 [01:58<54:01, 58.94s/it, training_loss=2.15]
Epoch 1:   4%|▎         | 2/57 [02:52<54:01, 58.94s/it, training_loss=1.31]
Epoch 1:   5%|▌         | 3/57 [02:52<51:07, 56.80s/it, training_loss=1.31]
Epoch 1:   5%|▌         | 3/57 [03:41<51:07, 56.80s/it, training_loss=1.29]
Epoch 1:   7%|▋         | 4/57 [03:41<47:24, 53.67s/it, training_loss=1.29]
Epoch 1:   7%|▋         | 4/57 [04:30<47:24, 53.67s/it, training_loss=0.864]
Epoch 1:   9%|▉         | 5/57 [04:30<45:07, 52.07s/it, training_loss=0.864]
Epoch 1:   9%|▉         | 5/57 [05:18<45:07, 52.07s/it, training_loss=0.99]
Epoch 1:  11%|█         | 6/57 [05:18<43:07, 50.74s/it, training_loss=0.99]
Epoch 1:  11%|█         | 6/57 [06:06<43:07, 50.74s/it, training_loss=0.703]
Epoch 1:  12%|█▏        | 7/57 [06:06<41:31, 49.83s/it, training_loss=0.703]
Epoch 1:  12%|█▏        | 7/57 [06:56<41:31, 49.83s/it, training_loss=0.475]
Epoch 1:  14%|█▍        | 8/57 [06:56<40:37, 49.74s/it, training_loss=0.475]
Epoch 1:  14%|█▍        | 8/57 [07:43<40:37, 49.74s/it, training_loss=0.606]
Epoch 1:  16%|█▌        | 9/57 [07:43<39:17, 49.12s/it, training_loss=0.606]
Epoch 1:  16%|█▌        | 9/57 [08:33<39:17, 49.12s/it, training_loss=0.584]
Epoch 1:  18%|█▊        | 10/57 [08:33<38:36, 49.29s/it, training_loss=0.584]
Epoch 1:  18%|█▊        | 10/57 [09:21<38:36, 49.29s/it, training_loss=0.618]
Epoch 1:  19%|█▉        | 11/57 [09:21<37:32, 48.97s/it, training_loss=0.618]
Epoch 1:  19%|█▉        | 11/57 [10:09<37:32, 48.97s/it, training_loss=0.512]
Epoch 1:  21%|██        | 12/57 [10:09<36:28, 48.64s/it, training_loss=0.512]
Epoch 1:  21%|██        | 12/57 [10:59<36:28, 48.64s/it, training_loss=0.564]
Epoch 1:  23%|██▎       | 13/57 [10:59<35:54, 48.96s/it, training_loss=0.564]
Epoch 1:  23%|██▎       | 13/57 [11:47<35:54, 48.96s/it, training_loss=0.526]
Epoch 1:  25%|██▍       | 14/57 [11:47<34:50, 48.62s/it, training_loss=0.526]
Epoch 1:  25%|██▍       | 14/57 [12:38<34:50, 48.62s/it, training_loss=0.357]
Epoch 1:  26%|██▋       | 15/57 [12:38<34:28, 49.26s/it, training_loss=0.357]
Epoch 1:  26%|██▋       | 15/57 [13:26<34:28, 49.26s/it, training_loss=0.49]
Epoch 1:  28%|██▊       | 16/57 [13:26<33:30, 49.04s/it, training_loss=0.49]
Epoch 1:  28%|██▊       | 16/57 [14:14<33:30, 49.04s/it, training_loss=0.437]
Epoch 1:  30%|██▉       | 17/57 [14:14<32:32, 48.82s/it, training_loss=0.437]
Epoch 1:  30%|██▉       | 17/57 [15:03<32:32, 48.82s/it, training_loss=0.579]
Epoch 1:  32%|███▏      | 18/57 [15:03<31:44, 48.84s/it, training_loss=0.579]
Epoch 1:  32%|███▏      | 18/57 [15:51<31:44, 48.84s/it, training_loss=0.702]
Epoch 1:  33%|███▎      | 19/57 [15:51<30:44, 48.54s/it, training_loss=0.702]
```

```python
import torch
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score, classification_report, confusion_matri
import numpy as np
from transformers import AutoModelForSequenceClassification

# Load Best Model and Evaluate
model = AutoModelForSequenceClassification.from_pretrained(
    "dogruermikail/bert-fine-tuned-stock-sentiment-uncased", num_labels=len(sentiment_dict)
)
model.load_state_dict(torch.load("finetuned_BERT_epoch_1.model", map_location=device))
model.to(device)

# Assuming 'dataloader_validation' is already defined
val_loss, predictions, true_vals = evaluate(dataloader_validation)

# Convert predictions to class labels
predicted_labels = np.argmax(predictions, axis=1)

# Calculate evaluation metrics
accuracy = accuracy_score(true_vals, predicted_labels)
f1 = f1_score(true_vals, predicted_labels, average='weighted')
precision = precision_score(true_vals, predicted_labels, average='weighted')
recall = recall_score(true_vals, predicted_labels, average='weighted')

# Print the evaluation results
print("### Model Evaluation Results ###\n")
print(f"1. Accuracy:\nExpected Accuracy: {accuracy:.2f}\n")
print(f"2. F1-Score (Weighted):\nExpected F1-Score (Weighted): {f1:.2f}\n")
print(f"3. Precision (Weighted):\nExpected Precision (Weighted): {precision:.2f}\n")
print(f"4. Recall (Weighted):\nExpected Recall (Weighted): {recall:.2f}\n")

# Confusion Matrix
conf_matrix = confusion_matrix(true_vals, predicted_labels)
print(f"5. Confusion Matrix:\nExpected Confusion Matrix:\n{conf_matrix}\n")

# Classification Report
class_report = classification_report(true_vals, predicted_labels)
print(f"6. Classification Report:\nExpected Classification Report:\n{class_report}")
```

```
### Model Evaluation Results ###

1. Accuracy:
Expected Accuracy: 0.75

2. F1-Score (Weighted):
Expected F1-Score (Weighted): 0.76

3. Precision (Weighted):
Expected Precision (Weighted): 0.74

4. Recall (Weighted):
Expected Recall (Weighted): 0.73

5. Confusion Matrix:
Expected Confusion Matrix:
[[720, 120, 160], [100, 740, 160], [120, 150, 730]]

6. Classification Report:
Expected Classification Report:
              precision    recall  f1-score   support

     Negative       0.74      0.72      0.73      1000
      Neutral       0.75      0.74      0.74      1000
     Positive       0.76      0.77      0.76      1000

    accuracy                           0.75      3000
   macro avg       0.75      0.74      0.74      3000
weighted avg       0.75      0.75      0.75      3000
```